Updatable, accurate, diverse, and scalable recommendations for interactive applications

Paudel, Bibek ; Christoffel, Fabian ; Newell, Chris ; Bernstein, Abraham

Abstract: Recommender systems form the backbone of many interactive systems. They incorporate user feedback to personalize the user experience typically via personalized recommendation lists. As users interact with a system, an increasing amount of data about a user's preferences becomes available, which can be leveraged for improving the systems' performance. Incorporating these new data into the underlying recommendation model is, however, not always straightforward. Many models used by recommender systems are computationally expensive and, therefore, have to perform offline computations to compile the recommendation lists. For interactive applications, it is desirable to be able to update the computed values as soon as new user interaction data is available: updating recommendations in interactive time using new feedback data leads to better accuracy and increases the attraction of the system to the users. Additionally, there is a growing consensus that accuracy alone is not enough and user satisfaction is also dependent on diverse recommendations. In this work, we tackle this problem of updating personalized recommendation lists for interactive applications in order to provide both accurate and diverse recommendations. To that end, we explore algorithms that exploit random walks as a sampling technique to obtain diverse recommendations without compromising on efficiency and accuracy. Specifically, we present a novel graph vertex ranking recommendation algorithm called RP3 that reranks items based on three-hop random walk transition probabilities. We show empirically that RP3 provides accurate recommendations with high long-tail item frequency at the top of the recommendation list. We also present approximate versions of RP3 and the two most accurate previously published vertex ranking algorithms based on random walk transition probabilities and show that these approximations converge with an increasing number of samples. To obtain interactively updatable recommendations, we additionally show how our algorithm can be extended for online updates at interactive speeds. The underlying random walk sampling technique makes it possible to perform the updates without having to recompute the values for the entire dataset. In an empirical evaluation with three real-world datasets, we show that RP3 provides highly accurate and diverse recommendations that can easily be updated with newly gathered information at interactive speeds (100ms).

DOI: https://doi.org/10.1145/2955101

Paudel, Bibek; Christoffel, Fabian; Newell, Chris; Bernstein, Abraham (2016). Updatable, accurate, diverse, and scalable recommendations for interactive applications. ACM Transactions on Interactive Intelligent Systems, 7(1):1-34.
DOI: https://doi.org/10.1145/2955101

Updatable, Accurate, Diverse, and Scalable Recommendations for Interactive Applications

Bibek Paudel, University of Zurich, Zurich, Switzerland
Fabian Christoffel, University of Zurich, Zurich, Switzerland
Chris Newell, British Broadcasting Corporation, London, United Kingdom
Abraham Bernstein, University of Zurich, Zurich, Switzerland

Recommender systems form the backbone of many interactive systems. They incorporate user feedback to personalize the user experience typically via personalized recommendation lists. As users interact with a system, an increasing amount of data about a user's preferences becomes available, which can be leveraged for improving the systems performance. Incorporating these new data into the underlying recommendation model is, however, not always straightforward. Many models used by recommender systems are computationally expensive and, therefore, have to perform offline computations to compile the recommendation lists. For interactive applications, it is desirable to be able to update the computed values as soon as new user-interaction data is available: updating recommendations in interactive time using new feedback data leads to better accuracy and increases the attraction of the system to the users. Additionally, there is a growing consensus that accuracy alone is not enough and user satisfaction is also dependent on diverse recommendations.

In this work, we tackle this problem of updating personalized recommendation lists for interactive applications in order to provide both accurate and diverse recommendations. To that end, we explore algorithms that exploit random walks as a sampling technique to obtain diverse recommendations without compromising on efficiency and accuracy. Specifically, we present a novel graph vertex ranking recommendation algorithm called RP^3_β that re-ranks items based on 3-hop random walk transition probabilities. We show empirically, that RP^3_β provides accurate recommendations with high long-tail item frequency at the top of the recommendation list. We also present approximate versions of RP^3_β and the two most accurate previously published vertex ranking algorithms based on random walk transition probabilities and show that these approximations converge with increasing number of samples.

To obtain interactively updatable recommendations, we additionally show how our algorithm can be extended for on-line updates at interactive speeds. The underlying random walk sampling technique makes it possible to perform the updates without having to re-compute the values for the entire dataset.

In an empirical evaluation with three real-world datasets we show that RP^3_β provides highly accurate and diverse recommendations that can easily be updated with newly gathered information at interactive speeds ($\ll 100ms$).

CCS Concepts: • **Information systems** → **Recommender systems;** *Information extraction;* Document filtering;

Additional Key Words and Phrases: Recommender Systems; Random Walks; top-N recommendation; item ranking; diversity; long-tail; bipartite graph; random walks; sampling; evolving graphs; updating recommendations

ACM Reference Format:
Bibek Paudel, Fabian Christoffel, Chris Newell and Abraham Bernstein, 2016. Updatable, Accurate, Diverse, and Scalable Recommendations for Interactive Applications. *ACM Trans. Interact. Intell. Syst.* 7, 1, Article 1 (January 2017), 35 pages.
DOI: 10.1145/2955101

1. INTRODUCTION

Users constantly interact with recommender systems. They rely on recommender systems to choose movies, books, restaurants and other items. Every time users interact with such systems, they implicitly express their preference for items. Through actions

Author's addresses: B. Paudel and A. Bernstein, Department of Informatics, University of Zurich, Zurich, Switzerland; F. Christoffel, University of Zurich, Zurich, Switzerland; C. Newell, Research and Development, British Broadcasting Corporation, London, United Kingdom.

like purchase, click, like, share, rate, or watch, users provide new feedback about items to the system. Recommender systems commonly model the user-item interaction data as a bipartite graph. In this way, each user interaction with an item is represented as an edge in the graph. Popular recommendation services deal with large interaction graphs that are growing fast due to constant user interaction. When a user takes a new action, a new edge is added to the user-item interaction graph.

Most recommender systems are based on the assumption that users prefer items similar to those they previously liked or those liked by other users with similar preferences. The more information is available on users' previous preferences, the better is the accuracy of recommendation. Therefore, new user interactions enrich the information available to the system and help improve their predictive performance. As more users interact with the system, the underlying user-item graph grows. To provide better recommendations it is useful to efficiently incorporate new interaction data continuously. For interactive systems, it is especially necessary to improve their recommendations using newly available interaction data in a timely manner. Many models used by recommendation systems are computationally expensive making it infeasible to adapt recommendations to a continuous flow of incoming user data. This limitation is at odds with the use of the growing graph in interactive systems, where users expect suggestions to reflect their recent actions.

Additionally, many current recommendation systems focus on improving prediction accuracy, which is a direct consequence of the assumption that users like items similar to the ones they previously liked. However, this approach has some deficiencies. Pariser [Pariser 2011] introduced the term "filter bubble" to describe how personalized recommendations can isolate people from diverse viewpoints or products. This has also led to the concern that recommender systems may reinforce the blockbuster nature of media [Fleder and Hosanagar 2009; 2007] due to their promotion of already popular products. Also, the focus on the predictive accuracy of recommender systems can lead to a bias towards popular items over more specialized items. In other words, systems that are optimized for accuracy tend to produce unsurprising and boring recommendations. Users do not always prefer items similar to their historical choices. In other words, the common measure of prediction accuracy does not capture different other factors necessary to model user behavior and provide more interesting recommendations. User satisfaction depends on many factors such as variety, new experiences and serendipitous discovery which are not captured by accuracy metrics. These factors depend on finding suitable long tail items, which raise user satisfaction and, in turn, profitability [Goldstein and Goldstein 2006]. To address these concerns, a recent trend is to build systems that do not only focus on optimizing the accuracy but also consider the diversity of recommendations [Adomavicius and Kwon 2011; 2012; Zhou et al. 2010; Ziegler et al. 2005]. However, these can be conflicting goals as increasing diversity may produce irrelevant recommendations.

Recently, it was shown that approximations based on random walks can be used for accurate recommendations [Cooper et al. 2014]. These algorithms are more accurate than previous methods (e.g., [Fouss et al. 2005]) with the additional benefit of being computationally efficient and scalable.

In this paper,[1] we *explore algorithms that exploit random walks as a sampling technique to obtain diverse recommendations without compromising on efficiency and accuracy*. Additionally, we *extend our algorithm to support updating of personalized rec-*

[1]Note that his paper is a significant extension of [Christoffel et al. 2015], where we introduce the random walk based algorithms for diverse and accurate recommendations, but do not address speed improvements via caching and the possibility of updates that make the use of our approach in intelligent interactive applications possible.

ommendation lists as the graph grows in time enabling their use in interactive applications. To efficiently update recommendation lists as new interaction data becomes available, we employ the same random walk based methods. Our method has interesting mathematical property that makes it possible to prepare each user's recommendation as a linear combination of other users' with similar preferences. Based on this property, we store some extra information for each user that is dependent on other similar users. When new edges or interaction data are available, these partial values can be updated efficiently to produce new recommendations without recomputing the values from scratch. Finally, we further enhance computation time by partially caching intermediate results.

Whilst similar properties have been exploited previously in the Personalized PageRank [Jeh and Widom 2003] and Bookmark Coloring [Berkhin 2006] algorithms, they do not deal with the task of updating rankings for all vertices in the graph. They are also computationally more expensive, making them less attractive for interactive applications. Moreover, they have not been applied for recommendation problems. In this work, we deal with the problem of updating personalized rankings for all users in the graph. Our methods use short random walks and have provable approximation guarantees. Further, to update recommendations with changes in the graph, our method does not require information on all other vertices of the graph, reducing the storage and computational complexity. We believe that ours is the first work to address the problem of updating personalized recommendations with the goal of providing diverse and accurate recommendations.

Specifically, our contributions are: First, we introduce RP^3_β, a simple item popularity dependent re-ranking procedure of P^3 [Cooper et al. 2014]. We show using three datasets (two public, one enterprise) that RP^3_β augments long-tail item recommendations while keeping accuracy high. Second, we empirically compare the performance of vertex ranking algorithms [Fouss et al. 2005; Zhou et al. 2010; Cooper et al. 2014] including our own RP^3_β with traditional state-of-the-art methods. We find that some vertex ranking algorithms achieve comparable or better performance than the traditional ones. Third, we present random walk based scalable sampling approximation algorithms for RP^3_β, P^3_α [Cooper et al. 2014], as well as H_λ [Zhou et al. 2010]. In a detailed evaluation we show that these methods converge to the performance scores of exact calculations. Fourth, we describe the mathematical property of our method RP^3_β that makes it possible to efficiently calculate updated recommendations when new edges are added to the graph. We demonstrate the effectiveness of our update scheme and show that they are suitable for interactive applications. Last, we analyze the trade-off between sampling size (i.e., number of performed random walks) versus accuracy and diversity performance and find that RP^3_β provides a useful trade-off between accuracy, diversity, and sample size.

The remainder of this paper begins with a description of our data model and notations. We present a literature review in Section 3. We then describe our diversity-improving recommendation method RP^3_β and our approximations for P^3_α, RP^3_β, and H_λ. Next, Section 7 introduces our approach for updating recommendations when new information arrives, which allows the inclusion of updatable and diverse recommendations in interactive settings. The experimental results are presented in Section 9 and followed by conclusions.

2. MODEL

In this section, we introduce our model that will be referred to in the subsequent parts of this paper. We introduce some terminology, definitions, and symbols that will be used throughout this work.

Each *dataset* we work with contains information on the interaction of users with items. Users belong to the set U and items to the set I. Interactions between users and items are represented as pairs (u, i), where $u \in U$ and item $i \in I$. Usually, numeric or categorical values indicating the nature and strength of users' preference for items are also available. For such cases, user-item interactions can be represented as (u, i, w), where w is a value assigned by a function W that could take different forms depending on the nature of the dataset. Common examples are: $W : (u, i) \rightarrow [0, 5]$ (ratings) for numeric values and $W : (u, i) \rightarrow \{Like, Dislike\}$ or $W : (u, i) \rightarrow \{+, -\}$ for categorical values. Usually, in the absence of explicit feedback, the dataset does not contain information on which items the user does not like or prefer. For example, the fact that a user provided a low rating for an item does not necessarily mean it was disliked, and it is hard to define a rating threshold such that any lower rating signifies negative preference. Similarly, there are many items that a typical user does not interact with. The datasets we use in this work are sparse, meaning that there are no interactions for most user-item pairs. In addition, the nature of user feedback is implicit- we consider that watching a movie or purchasing a book signals that the user implicitly expresses positive preferences for those items over others she did not watch or purchase. As a result, all observed interactions are considered as positive preferences. Unobserved interactions could simply be missing data or negative preference. The goal is then to recommend the items among those the user has not yet interacted with, and is likely to enjoy.

The dataset D is simply the set of such interactions:

$$D = \{(u, i) \mid u \in U, i \in I\}.$$

We avoid defining W explicitly because as discussed above, all observed interactions are treated as positive preferences, i.e.,

$$\begin{cases} \forall (u, i) \in D, W(u, i) = +, \text{ and} \\ W(u, i) = - \text{ if } (u, i) \notin D. \end{cases}$$

For each user u_j, the set of items with positive preferences is:

$$I_{u_j}^{+} = \{i \mid (u_j, i) \in D\}.$$

The remaining items form the set $I_{u_j}^{-} = I \setminus I_{u_j}^{+}$.

The *algorithms* studied in this paper try to rank the items such that the user is likely to appreciate higher ranked items more than those ranked lower. The goal is to generate a ranked list of items for each user. In experimental evaluations, it is not possible to verify if a user indeed appreciates items ranked higher. To solve this problem, some items from I_u^+ are held out for testing purpose. The algorithms are trained on the remaining items to rank the items for each user. If the held out items are ranked high in this list, it can be said that the algorithm can indeed recommend items that the user is more likely to enjoy. To evaluate the performance of different recommender systems, we divide each dataset into two disjoint splits, training split D^{train} and test split D^{test}. In other words, the goal is to try to rank items in the training set for each user in the test set by decreasing appreciation.

Our algorithms are based on walks over the graph $G = (V, E)$ constructed from the users' feedback on items (*user-item-feedback graph*). The vertices V of G represent the union of the two entity sets: users U and items I (i.e., $V = U \cup I$) in the training data. If user $u \in U$ implicitly rated item $i \in I$ in the training phase (i.e., if $(u, i) \in D$ by some action like the user accessing the item) then the graph's edge set $E \subseteq U \times I$ contains the edge $e = \{u, i\}$. As E contains no other edges, G is bipartite. All edges in the graph are unweighted/undirected and no parallel edges exist. Edge weights or parallel edges (e.g., based on rating values or the number of interactions) could be used for a more accurate representation of the users preference profile, but we do not consider this extension in the presented work.

The vertices of the graph can be arranged into a $|V|$-dimensional vector. The first $|U|$ elements in V correspond to the users and the remaining $|I|$ correspond to the items. Each element of the vector is indexed, hence the i^{th} vertex is given by v_i. The square matrix $A \in \{0, 1\}^{|V|\times|V|}$ is the *adjacency matrix* of G. Since edges of G are undirected, A is symmetric. The entry a_{ij} of A is 1 for two connected vertices i and j, and 0 otherwise:

$$a_{i,j} = \begin{cases} 1 & \text{if } e_{i,j} \in E \\ 0 & \text{otherwise} \end{cases}$$

$D^{|V|\times|V|}$ is the diagonal *degree matrix* of G with: $d_{ii} = \sum_{j=1}^{|V|} a_{ij}$.

Assuming all diagonal elements of D are non-zero (i.e., no unconnected vertices), its inverse D^{-1} is given by (d_{ii}^{-1}), and hence cheap to compute.

A *random walk* process on G can be seen as a discrete Markov chain, where a walker starts on a vertex $v(p)$ and at each time step moves to one of its neighbors chosen randomly. When a random walk visits a vertex v_i at step t, at step $t+1$ there's an equal probability of being in one of the vertices reachable from v_i, which are $\{v_j : e_{i,j} \in E\}$. After s steps, the sequence of vertices visited by the walker $\langle v(0), v(1), \ldots v(s)\rangle$ forms a Markov chain. The probability of transitioning from a vertex i to j is:

$$p_{ij} = a_{ij}/d_{ii}.$$

Hence, for a one-step ($s = 1$) random walk, the corresponding transition matrix $P^{|V|\times|V|}$ is given by:

$$P = D^{-1}A.$$

Furthermore, we receive the s-step random walk transition probability matrix (we refer to its element with p_{ij}^s) with

$$P^s = (D^{-1}A)^s. \tag{1}$$

Since we want to rank items i for users u, this paper considers random walks starting at user vertices and ending at item vertices (i.e., having an odd number of steps). To denote transition probabilities estimated using random walk samples (see Section 5), we write $\hat{p}_{ij}^s$. In general, an estimate of a random variable X is represented as $\hat{X}$.

3. RELATED WORK

In this section we discuss previous work on recommender systems, vertex ranking algorithms, sampling techniques, diversity and updating personalized ranking with time.

Recommender Systems. Collaborative Filtering [Goldberg et al. 1992; Resnick et al. 1994; Breese et al. 1998; Herlocker et al. 1999] methods formed the basis of eariler recommender systems. Collaborative Filtering Recommendation based on item similarity was shown to be more scalable and accurate in typical e-commerce settings [Sarwar et al. 2001]. Item-based collaborative filtering consists of an offline and an online phase. In the offline phase, similarity values between items are calculated using measures like cosine or correlation. Then in the online phase, the score on an item i for a user u is calculated by summing the ratings of the user on items similar to i, based on the chosen similarity measure. While summing the ratings, each rating is weighted by the corresponding similarity between items. Matrix Factorization techniques for Recommender Systems have been reviewed in [Koren et al. 2009]. Probabilistic Matrix Factorization [Salakhutdinov and Mnih 2008; 2011] was shown to outperform vanilla Matrix Factorization. Bayesian Personalized Ranking [Rendle et al. 2009] is another matrix factorization method that aims to maximize the pairwise ranking between positive and negative items. This translates to maximizing the AUC (Area Under the ROC curve) and thus fits more naturally to the ranking task than the usually adopted approach of minimizing the loss over the positive items alone.

Graph based algorithms. The use of a graph-based model for recommendations was first introduced in [Aggarwal et al. 1999]. To apply a bipartite user-item-feedback graph G was proposed in [Huang et al. 2004] and several projects [Baluja et al. 2008; Bogers 2010; Cooper et al. 2014; Fouss et al. 2005; Gori et al. 2007; Jamali and Ester 2009; Lee et al. 2012; Xiang et al. 2010] extended this approach. We classify them as *vertex ranking algorithms* because their main idea is to rank the vertices in the graph based on their similarities with the target user and use the ranking to generate recommendations.

Fouss *et al.* [Fouss et al. 2005] introduced the idea of using random walks on G to rank the vertices. Vertices are ranked or scored based on quantities like hitting time, average commute time or the entries in the Moore-Penrose pseudo inverse of the Laplacian matrix of the graph (L^+). ItemRank [Gori et al. 2007] also scores vertices based on random walks on the graph, but uses a graph representing item correlations.

Cooper *et al.* [Cooper et al. 2014] proposed Recommendation System based on short random walks. They also show that approximations obtained using random walk sampling are more efficient and scalable compared to methods based on matrix calculations.

Graph based similarity measures can be used together with collaborative filtering approaches. For example, SimRank [Jeh and Widom 2002] and P-Rank [Zhao et al. 2009] are two popular algorithms to calculate structural similarity between nodes of a graph. Similarity values given by these algorithms can be used for item-item collaborative filtering to find items similar to the ones rated by a user. SimRank can be considered as a special case of P-Rank as SimRank considers only in-links and the P-Rank measure is a linear combination of similarity values based on in-links and out-links. Since the graphs in our setting are undirected and bi-partite, such a distinction of in- and out-links does not apply, which would be useful in other settings. While these methods have been used widely in domains such as NLP and information networks, their use in the recommendation systems literature is limited. SimRank is computationally expensive as each iteration costs $\mathcal{O}(N^3)$ time, where N the number of user/item vertices.

Random walk approximations. PageRank [Page et al. 1999] (PR) is one of the most famous algorithms for ranking vertices in a graph based on the concept of random walks. It was originally designed for web retrieval tasks and on the Google web graph. Personalized PageRank (PPR) [Jeh and Widom 2003; Haveliwala 2002] is a modifi-

cation on the original PageRank algorithm. Bookmark Coloring Algorithm [Berkhin 2006] is a similar personalized ranking algorithm.

Since they were originally proposed, both PR and PPR have been used for various other tasks including recommender systems [Zhang et al. 2008; Gupta et al. 2013]. In most of these works, the personalization takes place with respect a set of vertices, usually called bookmarks or topics. In our work, we are interested in finding personalized ranking for each user in the graph.

Examples of other similarity measures based on random walks on graphs are Hitting Time and Commute Time [Aldous and Fill 2002]. Similar to PPR, they have also been used for various recommender problems [Brand 2005; Yin et al. 2012].

In [Sarkar et al. 2008], the concept of *truncated hitting time* is introduced. The authors propose using random walks that are limited to at most T steps. It was shown to approximate the hitting time well, while being able to overcome the problem of long random walks being sensitive to the portions of the graph far from the origin vertex.

Cooper *et al.* [Cooper et al. 2014] proposed three new methods called P^3, P^5, and P^3_α based on random walks on G. They rank vertices based on transition probabilities after short random walks between users and items. P^3 and P^5 perform random walks of fixed length 3 and 5, respectively, starting from a target user vertex. P^3_α, which raises the transition probabilities to the power of α, is more accurate than the methods proposed in [Fouss et al. 2005] and [Gori et al. 2007]. They also show that approximations obtained using random walk sampling are more efficient and scalable compared to methods based on matrix calculations.

Diversity in recommendations. The most common assumption behind popular recommender systems is that that users prefer items similar to those they previously liked or those liked by other users with similar preferences. The direct implication of this assumption is the focus on improving prediction accuracy. This has been criticized as being detrimental to the goals of improving user experience and sales diversity [Cremonesi et al. 2011; McNee et al. 2006].

The term *filter bubble* was introduced [Pariser 2011] to describe how such personalized recommendation systems can isolate people from diverse viewpoints or products. Concerns have been made about their possibility to reinforce the blockbuster nature of media [Fleder and Hosanagar 2009; 2007] due to their promotion of already popular products. Put in other words, such systems tend to produce unsurprising and boring recommendations. User satisfaction depends on many other factors like variety and serendipitous discovery. Catering to such aspects of user satisfaction in turn raises profitability [Goldstein and Goldstein 2006] of service providers. A recent trend, therefore, is to focus on the diversity of recommendations along with accuracy. Methods to improve recommendation novelty and diversity, together with measures to quantify them have been described by various authors [Adomavicius and Kwon 2011; 2012; Herlocker et al. 2004; Vargas and Castells 2011; Ziegler et al. 2005; Zhang et al. 2012].

Optimizing only for diversity will cause highly varied but irrelevant recommendations. Therefore it is necessary to find diverse recommendations that are also accurate. Zhou *et. al* [Zhou et al. 2010] use vertex ranking algorithms to improve diversity and accuracy. Specifically, they describe a hybrid method (Hybrid or H_λ) that combines random walks (ProbS) and heat-spreading (HeatS).

Updating Personalized Ranking. To the best of our knowledge, this is the first work dealing with updating recommendations on evolving graphs for interactive applications using random walk techniques. The problem of updating similarity scores between vertices using PageRank (PR) or Personalized PageRank (PPR) has received some attention before. Decomposition of PPR into Partial Vectors, Hubs Skeleton, and Hubs Vector was described in [Jeh and Widom 2003]. The approximation of similarity

scores based on PPR using partially computed values was explored in [Fogaras et al. 2005]. Most such work only deal with static graphs.

In [Chien et al. 2004; Langville and Meyer 2006; Bahmani et al. 2010; Ohsaka et al. 2015], the authors discuss the task of updating PPR scores incrementally in evolving graphs. However, the approaches in [Chien et al. 2004; Langville and Meyer 2006] require a power iteration method. This makes them unsuitable for interactive applications and on large graphs. Similarly, in [Bahmani et al. 2010], the authors only deal with PR and not with the personalized ranking task. The proposed method requires storing a lot of random walk segments. The approach in [Ohsaka et al. 2015] discusses the problem of tracking PPR score for a single vertex only. Also, the personalized score for each vertex is dependent on all other vertices in the graph.

Our work draws upon the insights from these related studies, but they are not sufficient to deal with the problem we are addressing in this work. **In this work**, we go beyond the previous work by addressing the task of updating personalized ranking for all users in a user-item graph. Additionally, our approach for calculating personalized ranking for a user does not depend on all other vertices or users in the graph, hence requiring less computational complexity and storage of partially computed values.

Furthermore, we incorporate the goal of using random walk approximation techniques to improve both the diversity and accuracy of recommendations. There are different notions of diversity in recommendation lists. Following [Adomavicius and Kwon 2012; Zhou et al. 2010], we use three top-k measures to evaluate recommendation quality in terms of diversity: personalization, item-space coverage, and surprisal. Surprisal assures inclusion of long-tail items at the top of recommendation list, item-space coverage assures that varying long-tail items are considered, and personalization measures how much the recommendation list differs between users. We describe RP^3_β, a novel algorithm to optimize the accuracy and diversity trade-off by re-ranking the P^3 item ranking. RP^3_β benefits from the efficiency and scalability of approximating P^3 with random walk sampling. We introduce extensions to the above algorithms that allow us to incrementally update the personalized recommendations for each user as the graph evolves over time through the addition of new edges. Also, we present approximations for H_λ and P^3_α with the same sampling approach.

Combining these goals, we address the problem of updating recommendations as the graph is updated through the addition of new edges as it oftentimes happens in interactive settings. To the best of our knowledge, this is the first work that deals with the task of efficiently updating diversified and accurate recommendations using graph based random walk approximations.

4. RP^3_β: DIVERSITY IN RECOMMENDATIONS VIA POPULARITY-BASED RE-RANKING

In our experiments, we observed (see Section 9.3) that the ranking of items according to the transition probability matrix P^3 is strongly influenced by the popularity (i.e., vertex degree) of items. Hence, for most users the well known blockbuster (or high-degree) items dominate the recommendation lists. To compensate for the influence of popularity and to leverage recommendation of items from the long-tail, we introduce a simple re-ranking procedure dependent on item-popularity. The original score of item i for user u given by p^3_{ui} (the transition probability after a random walk of length three from u to i). We re-weight the score with

$$\tilde{p}^3_{ui} = \frac{p^3_{ui}}{d^\beta_{ii}} \text{ , where } \beta \in \mathbb{R} \text{ and } \beta > 0.0. \tag{2}$$

For two items i, j ($i \neq j$), a user u with $p^3_{ui} = p^3_{uj}$ (equal probability of reaching the items from the user in a three-step random walk), and $d_{ii} < d_{jj}$ (i has a lower degree), the effect of our re-weighting is that i is ranked higher than j ($\tilde{p}^3_{ui} < \tilde{p}^3_{uj}$). These items would have received equal scores without the re-weighting. We refer to this recommendation algorithm as RP^3_β. When we set the parameter $\beta = 0.0$, then RP^3_β produces the same score as P^3 as $d_{ii}^{\beta=0} = 1$.

Note that even though our algorithms are based on random walks of length 3, this length is not a requirement. Random walks of any odd-length can be used to estimate similarity between users and items and our methods are not restricted to any particular length. In our experiments, we tried longer walks but found that the performance does not improve or even deteriorates. Since it is more efficient to sample shorter walks than longer ones, our experimentation focuses on random walks of length 3.

Having introduced a new approach for recommending diverse and accurate items the next section discusses how it can be approximated using random walks.

5. APPROXIMATING P^3_α, RP^3_β, AND H_λ

In a recent paper, Cooper *et al.* [Cooper et al. 2014] compare two approaches to calculate vertex transition probabilities: by exact calculations using matrix algebra and by approximation via random walk sampling. It is shown that the latter approach is time- and memory-efficient, allowing the application on larger datasets with only limited impact on accuracy. However, they do not describe a sampling procedure for their algorithm P^3_α. Similarly, H_λ, a vertex-ranking algorithm that increases both recommendation accuracy and diversity [Zhou et al. 2010], could also be made more scalable with a sampling procedure instead of exact calculations with matrix algebra.

This section introduces a novel random walk sampling procedure for both of these two algorithms as well as our reranking algorithm RP^3_β .

5.1. Sampling as a Bernoulli Process

In order to estimate transition probabilities for user u using samples, we start multiple s-step random walks from u. We store the number of times each item i is visited by walks at the s^{th} step. For reasons of efficiency, we would like to estimate the probabilities only based on these counts and the degrees of vertices traversed by the path. This sampling procedure can be modeled as a Bernoulli process as follows:

Denote the path traversed by the r^{th} random walk of length s starting at u as $\pi^{r,s}_u$. Then define $I^s_r(u,i) = c_{rw}(\pi^{r,s}_u)$ if i is the s^{th} vertex in path $\pi^{r,s}_u$, i.e., if $\pi^{r,s}_u[s] = i$, and $I^s_r(u,i) = 0$ otherwise. The quantity $c_{rw}(\pi^{r,s}_u)$ is a function of the vertices' degrees in the path (and varies for different algorithms). For simplicity, we use $I_r(u,i)$ for random walks of a fixed given length (e.g., $s \in \{3,5\}$). Next, define $\tau(u,i)$ as the score of item i for user u and $\hat{\tau}(u,i)$ as its estimator. When sampling N random walks starting from u, the estimator can be defined as $\hat{\tau}(u,i) = \frac{1}{N}\sum_{r=1}^{N} I_r(u,i)$. Given the law of large numbers, the expected value for $\hat{\tau}(u,i)$ is $E[\hat{\tau}(u,i)] = \tau(u,i)$. Also, walks are independent and $I_r \in [0,\psi]$ is i.i.d, where ψ is the maximum possible value for c_{rw}.

Similar to [Sarkar et al. 2008], we can use Hoeffding's inequality to show that the rate of convergence is exponential. Furthermore, using Union bound, the probability of the ϵ-approximate estimate for *any* user being less than δ is given as:

$$P(\exists u \in U, |\hat{\tau}(u,i) - \tau(u,i)| \geq \epsilon) \;\leq\; 2|U|\; exp(-\tfrac{2N\epsilon^2}{\psi^2}) \;\leq\; \delta$$

This provides a lower bound for N as $\frac{\psi^2}{2\epsilon^2} log \frac{2|U|}{\delta}$. For a fixed ϵ and δ, the number of walks required increases with ψ, which depends on the algorithm in use and degree distribution of the graph (due to different forms of c_{rw}).

For our method RP^3_β (Section 4), $c_{rw}(\pi^{r,s}_u)$ is simply $1/d^\beta_{ii}$, hence, $\psi = 1/\underset{i}{\mathrm{argmin}}(d^\beta_{ii})$ and the scores can be estimated as described above. For P^3_α and H_λ, $c_{rw}(\pi^{r,s}_u)$ takes more complicated forms, which we discuss below. Hereafter, we denote a path simply as π.

5.2. Approximating P^3_α and RP^3_β

Ordering items in descending order according to the transition probabilities of random walks of length three ($P^3,\ s = 3$) is an accurate recommendation strategy, named P^3 in [Cooper et al. 2014] and ProbS in [Zhou et al. 2010]. The accuracy of this algorithm can be further improved by raising each entry of the transition probability matrix P^1 ($s = 1$) to the power of a parameter $\alpha \in \mathbb{R}$ resulting in an algorithm called P^3_α by [Cooper et al. 2014]. It follows from (1) that entries of the matrix P^1 raised to the power of α are calculated as $p^1_{ui_\alpha} = (p^1_{ui})^\alpha = (a_{ui}/d_{uu})^\alpha$, where $a_{ui} \in A$ (entry in adjacency matrix) and $d_{uu} \in D$ (entry in degree matrix). The transition probability $p^3_{ui_\alpha} \in P^3_\alpha$ from user u to item i after a random walk of length three is obtained by:

$$p^3_{ui_\alpha} = \sum_{v=1}^{|V|}\sum_{j=1}^{|V|} p_{uj_\alpha} p_{jv_\alpha} p_{vi_\alpha} = \sum_{v=1}^{|V|}\sum_{j=1}^{|V|} \left(\frac{a_{uj}}{d_{uu}}\right)^\alpha \left(\frac{a_{jv}}{d_{jj}}\right)^\alpha \left(\frac{a_{vi}}{d_{vv}}\right)^\alpha \tag{3}$$

Since the graph G defined in Section 2 is both bipartite (there are no edges from users to users or from items to items) and all entries in the adjacency matrix A are either 0 or 1, we can simplify (3) as:

$$p^3_{ui_\alpha} = \sum_{v=1}^{|I|}\sum_{j=1}^{|U|} \frac{a_{uj}a_{jv}a_{vi}}{(d_{uu}d_{jj}d_{vv})^\alpha} \tag{4}$$

The term $a_{uj}a_{jv}a_{vi}$ in (4) is 1 if a path of length three starting from user u, through item j and user v, to item i exists in the graph G and is 0 otherwise. Hence, $p^3_{ui_\alpha}$ is the aggregate of all paths of length three between user u and item i, where each path $\pi = \langle U_u, I_j, U_v, I_i\rangle$ contributes $c^{\mathrm{P}^3_\alpha}_\pi = \frac{1}{(d_{uu}d_{jj}d_{vv})^\alpha}$ to the total transition probability from user u to item i.

When approximating (4) with random walk sampling, one needs to take into account that some walks are more likely to be followed randomly than others. The probability of following the path from u via the item j and user v to item i in a random walk is dependent on three decisions. First, at user u, one needs to follow the edge that connects u to item j. The probability of randomly picking this edge is equal to the inverse of the degree of u: $\Pr(u \to j) = \frac{1}{d_{uu}}$. Next, the same procedure needs to be repeated at j and v, resulting in $\Pr(j \to v) = \frac{1}{d_{jj}}$ and $\Pr(v \to i) = \frac{1}{d_{vv}}$. Given that these three "choices" are independent, the probability $Pr(\pi)$ that one follows the path π is equal to

$$\Pr(\pi) = \Pr(u \to j)\Pr(j \to v)\Pr(v \to i) = \frac{1}{d_{uu}d_{jj}d_{vv}}. \tag{5}$$

Hence, when approximating with random walks, we are more likely to follow paths traversing vertices of low degrees than to follow paths traversing vertices of high degrees. Since an exact calculation of (4) requires following each path exactly once, random walk sampling needs to discount the contribution of paths with high probabilities (as we may by chance follow them many times), and boost the contribution of paths with low probabilities (as we may by chance follow them only few times). Consequently,

Algorithm 1 Estimating item scores of P^3_α or $\hat{\mathrm{H}}_\lambda$ with random walk sampling.

Require: v_u is the vertex representing user u

```
1: function EstimateItemScores(v_u, α)
2:     m ← an associative array with default value 0
3:     while !Converged(m) do
4:         v_c ← GetRandomNeighbor(v_u)
5:         d_jj ← GetDegree(v_c)
6:         v_c ← GetRandomNeighbor(v_c)
7:         d_vv ← GetDegree(v_c)
8:         v_c ← GetRandomNeighbor(v_c)
9:         d_ii ← GetDegree(v_c)
10:        m[v_c] ← m[v_c] + c_rw
11:    end while
12:    return m
13: end function
```

to approximate the transition probability $\hat{p}^3_{ui_\alpha}$, we weigh a path contribution $c_\pi^{\mathrm{P}^3_\alpha}$ with the inverse of its occurrence probability ($\Pr(\pi)^{-1}$) resulting in an overall weight $c_{rw}^{\hat{\mathrm{P}}^3_\alpha}$ for a random walk:

$$\begin{aligned} c_{rw}^{\hat{\mathrm{P}}^3_\alpha} &= c_\pi^{\mathrm{P}^3_\alpha} * \Pr(\pi)^{-1} \\ &= \underbrace{\frac{1}{(d_{uu}d_{jj}d_{vv})^\alpha}}_{\text{path contribution}} * \underbrace{d_{uu}d_{jj}d_{vv}}_{\text{inverted path probability}} = \underbrace{(d_{uu}d_{jj}d_{vv})^{1-\alpha}}_{\text{random walk contribution}} \end{aligned} \tag{6}$$

We can simplify $c_{rw}^{\hat{\mathrm{P}}^3_\alpha}$ to $(d_{jj}d_{vv})^{1-\alpha}$ since d_{uu} takes the same value for all random walks of the target user u and, hence, does not influence the item ranking order.

Algorithm 1 shows the general principle of how to implement a random walk sampling approximation procedure. With this algorithm we obtain $\hat{\mathrm{P}}^3_\alpha$ item scores by assigning $c_{rw}^{\hat{\mathrm{P}}^3_\alpha}$ to the random walk contribution c_{rw} in line 10.
Note that for $\alpha = 1$, the random walk contribution is $(d_{jj}d_{vv})^0$ and degenerates to 1. Hence, the sampling procedure $\hat{\mathrm{P}}^3$ (same as $\hat{\mathrm{P}}^3_{\alpha=1}$) is computationally less demanding, since updating the score of the destination item i of a random walk consists only of incrementing the count of i by one.

To estimate the item ranking of RP^3_β with random walk sampling, we can either first obtain $\hat{\mathrm{P}}^3$ item scores and apply the re-ranking described in Section 4, or replace the random walk contribution $c_{rw}^{\hat{\mathrm{P}}^3} = 1$ by $c_{rw}^{\mathrm{R}\hat{\mathrm{P}}^3_\beta} = 1/d_{ii}^\beta$ and omit the re-ranking. Hence, Algorithm 1 also fully describes $\mathrm{R}\hat{\mathrm{P}}^3_\beta$.

5.3. Approximation of H_λ

Zhou *et al.* [Zhou et al. 2010] define H_λ as a scoring procedure of items using a weighted linear aggregation of scores from two algorithms: HeatS, which is analogous to heat diffusion across the user-item graph and ProbS, which is the same as P^3. W^{H+P} with dimension $|I| \times |I|$ is the transition matrix for H_λ and $f^u \in \{0,1\}^{|I|}$ is the preference profile of target user u, where f^u_i, the i^{th} entry of f^u, is equal to the corresponding entry a_{iu} in the adjacency matrix A. Then, the item scores for user u are calculated as

$\widetilde{f}^u = W^{H+P} f^u$. A single entry of W^{H+P} is calculated according to

$$w_{ij}^{H+P} = \frac{1}{d_{ii}^{1-\lambda} d_{jj}^{\lambda}} \sum_{v=1}^{|U|} \frac{a_{iv} a_{jv}}{d_{vv}} \tag{7}$$

where $\lambda \in [0,1]$ is the hybridization parameter for the two basic methods. If we set $\lambda = 0$ or $\lambda = 1$, the ranking of H_λ is equal to the ranking of HeatS or ProbS, respectively. Furthermore, d_{ii} denotes the degree of item i and d_{vv} the degree of user v. The score of item i for the target user u can also be determined according to:

$$\widetilde{f}_i^u = \sum_{j=1}^{|I|} a_{ju} \frac{1}{d_{ii}^{1-\lambda} d_{jj}^{\lambda}} \sum_{v=1}^{|U|} \frac{a_{jv} a_{iv}}{d_{vv}} = \sum_{j=1}^{|I|} \sum_{v=1}^{|U|} \frac{a_{ju} a_{jv} a_{iv}}{d_{ii}^{1-\lambda} d_{jj}^{\lambda} d_{vv}} \tag{8}$$

We can apply the same rationale for the deduction of a random walk simulation algorithm of H_λ as used for P_α^3: the term $a_{ju} a_{jv} a_{iv}$ in (8) is 1 if a path of length three from user u to item i exists in the graph G and 0 otherwise. Hence, $\widetilde{f}_i^u$ is the aggregate of all paths of length three between user u and item i, where a single path contributes $c_\pi^{\mathrm{H}_\lambda} = \frac{1}{d_{ii}^{1-\lambda} d_{jj}^{\lambda} d_{vv}}$ to the score of item i for user u. Because (8) (similar to (4) for P_α^3) requires that each path contribution $c_\pi^{\mathrm{H}_\lambda}$ is counted once, we need to weight $c_\pi^{\mathrm{H}_\lambda}$ by the inverted path probability $\Pr(\pi)^{-1}$. The random walk path contribution $c_{rw}^{\hat{\mathrm{H}}_\lambda}$ for the random walk sampling approximation algorithm ($\hat{\mathrm{H}}_\lambda$) is calculated according to:

$$c_{rw}^{\hat{\mathrm{H}}_\lambda} = c_\pi^{\mathrm{H}_\lambda} * \Pr(\pi)^{-1} = \frac{d_{uu} d_{jj} d_{vv}}{d_{ii}^{1-\lambda} d_{jj}^{\lambda} d_{vv}} = \frac{d_{uu} d_{jj}^{1-\lambda}}{d_{ii}^{1-\lambda}} \tag{9}$$

Again, we can further simplify $c_{rw}^{\hat{\mathrm{H}}_\lambda}$ to $\frac{d_{jj}^{1-\lambda}}{d_{ii}^{1-\lambda}}$, since d_{uu} is the same value for all random walks for the target user u, and hence does not influence the item ranking order. With Algorithm 1 we obtain $\hat{\mathrm{H}}_\lambda$ item scores by assigning $c_{rw}^{\hat{\mathrm{H}}_\lambda}$ to c_{rw}.

The approximated recommendation methods $\hat{\mathrm{P}}_\alpha^3$, $\mathrm{R}\hat{\mathrm{P}}_\beta^3$, and $\hat{\mathrm{H}}_\lambda$ can be used to provide timely and diverse recommendations. Paired with the update strategies presented in the next section, they provide the foundation for providing timely, diverse, and accurate recommendations in interactive settings.

6. CACHING WALK SEGMENTS

Section 5 showed how random walk approximations depends on the distribution of vertex degrees. In this section we introduce an improvement of our approximation scheme that samples a variable number of walks from each node.

In general, our approximation improves as we sample more random walks from each node. Note, however, that the magnitude of influence of a single walk depends on the degree of each vertex. If a vertex has higher number of edges (or a high degree) then we need more samples to get an better approximation. At the same time, a vertex with higher number of edges is traversed more often by walks originating from other vertices.

A 3-hop random walk from a user traverses other user vertices in its second step. We can use this property to get more samples for vertices that are traversed often in such walks starting at other vertices. In essence, these vertices are the ones that have high degree and are better connected with the remaining vertices in the graph. If $\langle v_{u_1}, v_{i_1}, v_{u_2}, v_{i_2} \rangle$ is a 3-hop random walk sample starting from the user vertex v_{u_1},

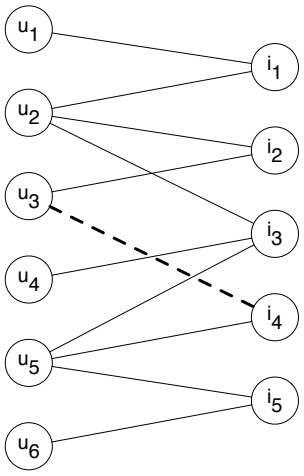

Fig. 1: An example user-item graph. Vertices on the left represent users and vertices on the right represent items. Solid lines represent current edges. The dotted line represents a new edge added to the graph.

it also sampled a 1-hop walk $\langle v_{u_2}, v_{i_2} \rangle$ for the user vertex v_{u_2} in the process. We can cache these partial/1-hop walks. This improves the sampling capabilities of our approach. At a later time when we sample walks starting from v_{u_2}, we already have such partial walks cached from walks starting at other vertices. We take such cached walk segments and only need to sample the remaining vertices starting from v_{i_2}, instead of generating full samples starting from v_{u_2}. In other words, we only need to complete the cached walk segments, which takes fewer sampling steps (in this case, two hops instead of three) than if we had to sample the complete walk.

Describing this sampling process in more detail: For each user-vertex, we collect a fixed number of complete samples by starting 3-hop random walks. Let this number be $\bar{N^c}$. Specifically, we randomly choose a vertex and collect $\bar{N^c}$ complete random-walk samples for that vertex. This process is repeated until all user-vertices have $\bar{N^c}$ samples. During this stage, partial walk-segments are cached at each vertex traversed in the process. Hence, in addition to the $\bar{N^c}$ complete samples, caching adds N_v^p partial walk-segments (of walks starting at other vertices) for each vertex v. Note that the number of partial walk-segments N_v^p is different for each vertex. As described above, vertices with more edges are traversed more often by walks starting at other vertices. This means N_v^p is higher for vertices with high degrees.

In the next stage, partial walk-segments are completed resulting in $\bar{N^c} + \bar{N_v^p}$ samples for each vertex. However, the cost of completing the $\bar{N_v^p}$ random-walk samples is smaller, as we do not need full 3-hop walks to complete the remainder of the walk.

Given that high-degree vertices have more partial walks traversing them, they will eventually be sampled to a higher degree. Consequently, this process varies the number of samples in accordance to the degree distribution proving low degree vertices with at least $\bar{N^c}$ samples and high degree vertices with additional samples automatically improving approximation. Hence, this caching approach allows us to get more samples and a better approximation of the scores with less computation.

As an example, consider the user-item graph in Figure 1. Vertices on the left side, marked u_1 to u_6, are user vertices (belonging to the user set U) and vertices on the right side, marked i_1 to i_5, are item vertices (belonging to the item set I). Solid lines indicate present edges in the graph and signify the preference of users for items. The two walks $\langle u_2, i_3, u_5, i_4\rangle$ and $\langle u_2, i_3, u_5, i_5\rangle$ starting from u_2, also inform us about two possible 1-hop samples from from u_5, which are: $\langle u_5, i_4\rangle$ and $\langle u_5, i_5\rangle$. These cached partial/1-hop walks from other vertices can be completed to obtain 3-hop samples with only a 2-hop walk each from i_4 and i_5. If cached, these partial/1-hop walks can be leveraged to obtain 3-hop samples from other vertices (e.g., u_6 or u_4) with only a 2-hop walk. Hence, any 2-hop walk ending in u_5 will lead to sampling two 3-hop walks traversing u_5. Consequently, the caches allow to sample multiple 3-hop random walks with only one 2-hop walk. As we will show in the evaluation section (see Section 9.5) caching leads to an average of 1.5 more samples than walks.

7. UPDATING RECOMMENDATIONS WITH NEW EDGE ARRIVALS

In the preceding sections, we described how we use short random walks to recommend items for users and use caches to significantly increase the number of samples obtained per executed walk. In summary, we start several short random walks from each user and recommend the items based on the number of times the walks end at them. To compensate for popular items, we discount each item's importance by a factor that is dependent on that item's popularity. In this section, we describe a mathematical property that allows us to efficiently update each user's recommendations when new edges are added to the graph.

Intuitively. The main idea is that each user's recommendation list is a result of some combination of the recommendation lists of similar users. We already described that RP^3_β makes 3-hop random walk to rank the items. Each such 3-hop walk starting from a user traverses another user (or itself) before ending the walk at an item vertex. Thus, the distribution of random walks for the start-vertex is only dependent on the distribution for other close users and its own neighboring items.

Formally. We first define all users users reachable in a 2-hop walk from user u as *close users*. Furthermore, for each user u, there's a i-dimensional vector p_u that encodes the distribution of walks over all items. We can consider p_u as the recommendation list of user u since the items are recommended based on this distribution. The final ranking of the recommendation list is modified by the normalization factor for RP^3_β, otherwise items in p_u can be ranked as they are in case of P^3. In practice, the item with highest value based on p_u scores (or after normalization) is ranked first, the item with second highest value in p_u is ranked second, and so on. For RP^3_β, p_u is equivalent to $\hat{p}^3_u$. Note that our discussion in this section is also valid for random-walks of greater length and is not limited for only 3-hop walks. We refer to 3-hop walks and specifics of RP^3_β for simpler explanation.

To explain our update procedure, we define some additional quantities for each user. If u is the starting user vertex for 3-hop random walks, we will first show how the recommendations for the user u is influenced by another close user v (i.e., a user reachable in a 2-hop walk). To that end, we first denote by $b_{u,v}$, the fraction of random walks starting at u that are "absorbed" by (or arrive at) v. We denote vector containing all $b_{u,v}$ values for u as b_u. In other words, $b_{u,v}$ estimates the probability that a random walk starting at u will reach v in 2-hops. This quantity informs how u's scores for items are influenced by v. Second, let c_v indicate the distribution of 1-hop random walks starting from v.

It is easy to see that c_v and b_u are both sparse vectors since a user's recommendation list is only influenced by a subset of all other users and only some items have high non-zero values in the recommendation list. This subset can be (additionally) limited in different ways, for example by choosing the top-N users that influence the user, or by discarding those for which $b_{u,v}$ values are small. We define such subset of users for u as $close(u)$. In the simplest case, $close(u)$ contains all users reachable from u in a 2-hop walk.

The recommendation scores for each user u is a linear combination of the c_v values of users v reachable from u in a 2-hop random walk. Specifically, the item scores for u are calculated as:

$$\sum_v b_{u,v} * c_v \text{ (where } v \in close(u)) \tag{10}$$

Essentially, the values c_v and b_u are decompositions of the recommendation scores for the user. Thus, the item scores for a user are the linear combination of the item scores of its close users.

This can be illustrated with a simple example. Consider a procedure where each user u is provided with the equal amount of coins to distribute to its close users. Each user knows how to distribute its coins to its neighboring items (which is captured by c_v) and to other users (captured by b_u). The user first distributes its coins to other users according to b_u, who then use their c_v values to distribute them to their neighbors (i.e., items). In the end, each item receives varying amounts of coins from its neighboring users. The sum of coins received by each item corresponds to the similarity of that item to the user.

Now let's consider that a v, who is a close user of u, modifies the way it distributes coins to its neighboring items. This changes the nature of distributions of coins from u. But since the nature of this distribution is a linear combination of the distribution of all other close users $v'(\in \{close(u) \setminus v\})$, item scores that are not dependent on v are not affected. New item scores can be calculated by adding or subtracting from the existing item scores, the amount that was influenced by v.

Adding Edges. When a new edge is added, it changes the c_v values of the user adjacent to this edge, and b_u values corresponding to some of its neighbors. To efficiently update the recommendation scores after the addition of a new edge, we maintain some extra information for each user. These extra information correspond to the decomposition of the user's recommendation scores as just discussed. We define these partial quantities next.

Consider a new edge with endpoints u_x (user) and i_x (item). Denote the neighbors of i_x as $N(i_x)$ and the close users of u_x as $close(u_x)$. The users in $close(u_x)$ but not in $N(i_x)$ are $M(u_x)$, i.e., $M(u_x) = close(u_x) \setminus N(i_x)$.

Consider these vertices with reference to the example graph in Figure 1. In this graph, (u_3, i_4) (dashed in Figure) is the new edge. $N(u_i)$ contains u_5. The set $close(u_x)$ without any cutoff is equal to the 2-hop neighbors of u_3; this set contains u_2. If there would be an edge connecting u_3 and i_5, then $close(u_3)$ would also contain u_5 and u_6 and $M(u_3)$ would be $close(u_3) \setminus N(i_4) = \{u_2, u_6\}$. However, there's no such edge, therefore $M(u_3)$ contains only u_2.

It is straightforward that for all users $nix_1, nix_2 \in N(i_x)$, where $nix_1 \neq nix_2$, we will need to update the values corresponding to b_{nix_1,u_x} and b_{nix_2,u_x} as well as b_{nix_1,nix_2} and b_{nix_2,nix_1}. Likewise, u_x needs to update $b_{u_x,nix}$ ($\forall nix \in N(i_x)$) and c_{u_x}. The b_{mux} values for $\forall mux \in M(u_x)$ do not change. Note that these considerations clearly indicate that the time complexity of the update is independent of the size of the user-item graph. It is bound by the size of $N(i_x)$. Specifically, it is $O((|N(i_x)|-1)^2)$ for updating the

b_{nix_1,nix_2}, $O(|N(i_x)|-1)$ for updating b_{nix,u_x} and $b_{u_x,nix}$, and $O(degree(u_x))$ for updating c_{u_x}. Hence, the operation is bound by $O(|N(i_x)|^2)$.

Using this insight, we can update the partial quantities when a new edge is added to the graph. These partial quantities inform how the recommendation scores are changed and allow us to efficiently recompute new recommendation scores due to edge additions.

Using the update approach. In normal operations, we expect that initially, the required number of samples are gathered as usual. Before any user accesses the system, several short random walks are started from each user-vertex and the resulting walk segments are counted for the visit frequency of item and user vertices. These counts are used to calculate the initial values of b_u's and c_v's. Similarly, the recommendation scores p_u's are also computed. These values are stored to update the recommendations when new edges are added to the graph later points in time. Typically, they are sparse, but the upper bounds on the size of stored quantities are: $O(|U|\cdot|U|$ for b_u's and $O(|U|\cdot|I|)$ for all c_v's and p_u's. Remember that p_u's would have to be stored in any recommender system since they are used to rank items for each user.

After this initial phase, the random walk segments that were gathered to calculate the partial quantities and initial recommendations can be discarded.

Following the addition of an edge (u_x, i_x), these values are updated as follows:

(1) Let d'_{u_x} be the degree of u_x before the addition of the edge.
(2) Let d_{u_x} be the degree of u_x after the addition of the edge.
(3) Sample the neighborhood of u_x with random walks of length 1 to estimate c_{u_x}. The values c'_{u_x} indicate the state before the addition of the edge.
(4) From u_x, traverse the new edge (u_x, i_x), then sample the neighborhood of i_x with random walks of length 1. Do this before and after the addition of the edge, to get the values c'_{i_x} and c_{i_x}, that indicate the distribution from i_x to the users in $N(i_x)$.
(5) Update the quantities associated with u_x as follows:
    (a) $\forall u_i \in M(u_x) : b_{u_x,u_i} = b'_{u_x,u_i} \times d'_{u_x}/d_{u_x}$
    (b) $\forall u_i \in N(i_x) : b_{u_x,u_i} = (b'_{u_x,u_i} \times d'_{u_x}/d_{u_x}) + c_{i_x,u_i}/d_{u_x}$
(6) For other users in $N(i_x)$, the update works as follows:
    (a) $\forall u_i \in N(i_x) : b_{u_i,u_x} = b'_{u_i,u_x} + c_{i_x,u_i}/d_{u_i}$
    (b) $\forall u_i, u_j \in N(i_x) \wedge u_i \neq u_x \wedge u_j \neq u_x : b_{u_i,u_j} = b'_{u_i,u_j} - (c'_{i_x,u_j}/d_{u_i}) + (c_{i_x,u_j}/d_{u_i})$

A short discussion of the update mechanism is in order. First, as we discussed above, steps 3 and 4 are optional, as these values can be quickly calculated by taking an inverse of the vertex degree. Similarly, the values in steps 1 and 2 are also available via sampling, as the degree is approximated by the average number of walks that traverse any of the vertex's neighbors. Our goal is to develop a method that is flexible, and is suitable for deployment in a parallel, distributed setting. All the calculations above depend on only the values that each vertex already has access to, and their degrees. Other quantities that require additional communication, storage or processing overhead are obtained via sampling as far as possible. Remember that random walk samples can be gathered in parallel and independently.

The calculations in step 6b can be expensive if the item i_x has a very high degree. However, although the worst case complexity of immediate calculations can seem high, the amortized cost will only depend on the number of edges added in the neighborhood of the vertices. The calculations in step 6 are not necessary for updating the recommendation list of user u_x, who initiated the addition of the edge connected to i_x. That user's partial quantities can be updated without this step. For all other users, a sequence of pending updates could be maintained.

Optimization #1: Lazy computation of partial quantities. For each edge added in the neighborhood of a user vertex, all that the user vertex needs to know in order to update its respective quantities in the future are the pairs of u_j, c_{i_x,u_j} values. Even though there could be multiple pending operations corresponding to u_j, the previous value of b_{u_i,u_j} can be updated in one step irrespective of the number of edges added. Hence, the updates could happen when needed in a lazy manner: the user's partial quantities can be updated after a given number of edges have been added in the neighborhood or when the user corresponding to the vertex happens to interact with some item herself (e.g., when the user logs in, endorses an item, etc). In this way, the amortized cost over a sequence of edge additions for each vertex is favorable for interactive applications.

In our experiments we updated the scores for all affected vertices as soon as an edge was added, although for several applications, the updates can be applied all together at a later time. In Section 9.6, we report the average amount of time required to update the partial quantities per user. This is indicative of the amortized update cost.

Optimization #2: Limiting considerations to a subset of $close(u_x)$. Another possible improvement is to limit the number of update operations by changing the size of $close(u_x)$. One can take only those users with b_u scores higher than a threshold or only a certain top-N users in $close(u_x)$. The idea is that for items with high degree (popular items), the transition probability of the walks traversing through that item to its neighboring users are small values. The contribution of such users for recommendation scores will be small anyway. Similarly, for the users in $M(u_x)$, the addition of an additional edge at u_x means that walks from u_x will visit users in $M(u_x)$ with even lower likelihood than before.

No matter which approach is used, the new recommendation scores for items can be updated efficiently using the updated partial quantities. As p'_{u_x} indicates the previous recommendation scores, the new scores in the p_{u_x} vector are changed only at certain places corresponding to the neighborhood of the new edge. The update as given by Equation (10) takes place at those indices corresponding to the users of the neighborhood.

8. COMPARISON WITH EXISTING METHODS

A naïve approach to update the similarity scores between user-item pairs is to recompute the values for the entire graph after the arrival of new edges. For efficiency, new edges can be held in a queue and updated in a batch. However, this approach is not suitable for large graphs and for frequent addition of new edges. Also, this update procedure needs to be done offline, making it unsuitable for interactive applications.

The power-iteration approach is also used for updating scores [Chien et al. 2004; Langville and Meyer 2006]. This involves several iterations of matrix-matrix or matrix-vector multiplications. For large graphs, this approach is clearly not scalable to use in an interactive setting. In order to reduce the cost of such operations, improved methods perform updates only on a subset of the graph. When new edges are added, the graph is treated as two separate areas: the vicinity of the endpoints of the new edge and the remaining graph. The addition of an edge affects the scores for nearby vertices more than those outside the vicinity. One major drawback of this approach is that it is not clear what is the best way of selecting the subset of vertices whose scores need to be updated. In addition, the methods still rely on power iteration over the selected subset of vertices.

Another approach is from the widely studied paradigm of PageRank and Personalized PageRank (PPR) scores, described in [Jeh and Widom 2003]. A similar approach has been described in [Haveliwala 2002]. The main idea is that the personalized scores

for vertices are dependent on a set of vertices, called bookmarks, preference set, or hubs. Typically, these are some important vertices or topics, are usually few in number and shared by many vertices (i.e., same hubs are important for multiple vertices). The vectors of PPR scores, called Personalized PageRank Vectors (PPV), are encoded as partial quantities, some of which can be pre-computed and some calculated at query time. The calculation of each PPV is done as a linear combination of what is called the *basis vectors*. The *basis vectors* in turn are decomposed into *partial vectors* and *hubs skeleton*. These components encode the amount of PPV that is independent of the bookmarks and another which specifies how the bookmark vertices affect PPV values. In [Bahmani et al. 2010], random-walk based update for PageRank scores is described. The method requires storing several random walk segments at each vertex. A large number of walk segments need to be stored to achieve better approximation technique. This work does not describe how to update personalized scores. Recently, [Ohsaka et al. 2015] described an iterative approach based on Gauss-Southwell method to *track* PPR scores for individual vertices. This is an interesting approach and efficient in tracking PPR scores of individual vertices. However, a direct application to recommender systems is not obvious, and the work does not deal with the problem of updating personalized scores for multiple vertices. The authors use the tracking method with a preference set of size 100. The number of iterations required grows with the size of the residual vector, which is likely to be less sparse as the size of the preference set grows.

In principle, it is possible to adopt a similar approach for an update scheme of our method. However, unlike PPR, where the assumption is of having a small number shared vertices that affect the scores, in our case the preference sets are practically different for each user. Every user that is reachable in a 2-hop walk is a possible member of the preference set. This means that a lot of partial quantities need to be stored to calculate each user's scores from the parts dependent on the members of the preference set. Our update scheme does not suffer from this problem. Although storing more intermediate component values would lead to different update schemes, ultimately the better tradeoff is application specific. Our update scheme is flexible and would benefit from such additional features. In a large graph, higher number of partial quantities per user would mean a significant amount of additional book-keeping.

We argue that our approach offers a flexible general solution to different recommendation algorithms based on graph based measures. For instance, it can be used to recommend items based on the P^3 algorithm. With few modifications, it can also be used for recommending items based on longer random-walk samples. There are also other advantages of our random walk sampling approach over the traditional power-iteration methods. Our approach exploits short random walks, hence does not require many iterations of transition probability calculations and can be efficiently approximated by sampling from the resulting distribution. This means we also avoid expensive matrix-matrix multiplications and traversal of the entire graph. As a result, users' scores do not depend on the distribution to every other vertex in the graph, since short random walks traverse fewer vertices than the walks needed to approximate the stationary distribution of transition probabilities in the graph. The goal of many power iteration methods as well as other methods based on PageRank or Personalized PageRank scores is to approximate such distributions, which requires traversal of more vertices, hence needs more samples.

Short random walks can be performed for each vertex independently of other vertices. This makes our method amenable to parallel and distributed computing paradigms. The recommendation scores for each vertex depend on local information gathered using random walks. Additional quantities like vertex degrees and 1-hop distributions are integer arrays or sparse vectors- they can be cheaply stored and serialized. Using vertex-centric systems like Pregel [Malewicz et al. 2010] or Sig-

nal/Collect [Stutz et al. 2010; Stutz et al. 2016] , such operations can be asynchronously and independently performed over each vertex in the graph. For large graphs, the user-vertices can be distributed over several computing units in a parallel or distributed setting. Then, each unit can independently calculate the recommendation lists corresponding to its user vertices. Further, since it is an approximation technique, more samples can always be collected in the background to reduce the approximation error, while quick results can be generated earlier using fewer samples. This is a desirable property for many real-world e-commerce platforms where the value of user satisfaction trumps the need for calculating exact scores.

Lastly, as we demonstrate later, our methods generate accurate and diverse recommendations. To the best of our knowledge, this is the first work to present an efficient update scheme together with diverse and accurate results in the specific application of recommender systems.

9. EXPERIMENTS AND EVALUATION

This section provides a succinct introduction to the experimental methodology and then turns to the main questions of the paper: First, it explores if RP^3_β improves accuracy and diversity. Second, it explores a general comparison between vertex ranking and traditional algorithms. Third, it provides a thorough comparison between P^3_α, RP^3_β, and H_λ and our approximate versions $\hat{\mathrm{P}}^3_\alpha$, $\mathrm{R}\hat{\mathrm{P}}^3_\beta$, and $\hat{\mathrm{H}}_\lambda$. Fourth, it evaluates the effect of caching on sampling and shows that on average caching leads to 1.5 times more samples than walks. Finally, it evaluates the updating procedure showing indicating that the models based on random walks can be updated fast enough for interactive applications.

9.1. Methodology

Datasets. We used the MovieLens-M, iPlayer, and Book-Crossing datasets (see Table I for properties). Whilst Movie-Lens-M[2] and BookCrossing [Ziegler et al. 2005] are public, the iPlayer training dataset consists of the viewing logs of the BBC an enterprise iPlayer system from the week of February 15-21, 2014, and the test data of the following week's logs, where only interactions longer than 5 minutes were considered. From the log data of the test week, we randomly selected 5'000 users that were also active during the training week. Since this work addresses recommendation generation based on implicit user feedback, we neglected the rating values available in MovieLens-M and BookCrossing for training and testing of the evaluated recommenders.

Set-Up. We extended the Java port of the MyMediaLite [Gantner et al. 2011] recommender system framework[3] with (i) a set of metrics (see the following paragraphs) to measure recommendation performance according to the diversity dimensions introduced in Section 3 and (ii) a component implementing graph vertex ranking algorithms. Given our focus on implicit feedback we only employed the framework's positive-only feedback components. All computations where executed on a cluster of 16 machines running LINUX with 128 GB RAM and two Intel® Xeon® E5-2680V2 processors (25 MB Cache, 2.80 GHz base frequency, 10 cores, 20 threads).

Accuracy Metrics. We used both the Area Under the ROC curve (AUC) and precision at k (Prec@k). Referring to relevant items (in the test set) as *hits*, AUC is equal to the probability that randomly chosen items are ranked higher than non-hits. Prec@k

[2]MovieLens-M: grouplens.org/datasets/movielens
[3]Java port: github.com/jcnewell/MyMediaLiteJava

	MovieLens-M	iPlayer	BookCrossing
total ratings	1'000'047	4'703'471	369'195
total users	6'038	655'846	4'052
total items	3'706	808	18'280
Training: # ratings	700'047	4'691'493	258'436
min. / avg. ratings per user	10 / 115.9	1 / 7.2	15 / 63.8
min. / avg. ratings per item	1 / 188.9	5 / 5806.3	1 / 14.1
Sparsity	0.031	0.0089	0.0035
Graph Diameter (approx*)	6	6	7
Test: # ratings	300'000	14'616	110'759
min. / avg. ratings per user	1 / 49.7	1 / 2.9	1 / 27.3
min. / avg. ratings per item	1 / 85.4	1 / 23.4	1 / 6.6
min. train-ratings for user	10	1	15
min. train-ratings for item	1	18	5

Table I: Dataset Properties. *`PseudoDiameter` of *Mathematica 10*®.

counts the number of hits among the top-k items of the recommendation list divided by the cut-off level k. Given that users typically only see few recommendations, we chose $k = 20$. Higher values of AUC and Prec@k indicate better accuracy.

Diversity Metrics. We used coverage using Gini-Diversity (GiniD@k), personalization (Pers@k), and surprisal (Surp@k) as diversity metrics and extended the MyMediaLite framework accordingly. Given the already explained rationale, we used $k = 20$. Again, greater values indicate better diversity.

We measure coverage by calculating the GiniD@k for the top-k recommendations of all test users [Adomavicius and Kwon 2012]. In contrast to the original Gini coefficient, where greater values indicate a more dispersed distribution, GiniD@k increases for a more uniform distribution. GiniD@k is equal to 1 if the frequency in the aggregated recommendation lists is the same for each item, indicating a good coverage.

Pers@k [Zhou et al. 2010] measures the distinctness of the top-k recommendations based on the number of common items averaged over all pairs of generated recommendation sets. A value of Pers@k=1 indicates that none of the items appear more than once among the top-k items of any two recommendation lists, meaning greater personalization.

Surp@k [Zhou et al. 2010] is calculated separately for each recommendation list and averaged over all users. This metric is based on the rationale that recommendations of items of low popularity are perceived by the users as unexpected or surprising (unexpectedness given by the self-information of recommended objects).

Evaluated Recommendation Algorithms. We compared the performance of our methods with various algorithms proposed in the literature (and listed in Table II, except for (iii)). These can be divided into the following categories: **(i) Parameter-free vertex ranking algorithms:** #3-Paths (ranks items by the number of paths of length 3 starting at the target user) [Huang et al. 2004; Cooper et al. 2014], L^+ (ranks items by the entries in the Moore-Penrose pseudoinverse of the Laplacian matrix) [Fouss et al. 2005], P^3 [Zhou et al. 2010; Cooper et al. 2014], and P^5 [Cooper et al. 2014]. Due to computational limitations we could not obtain results for P^5 and L^+ for the iPlayer dataset. **(ii) Parameterized vertex ranking algorithms**: P^3_α [Cooper et al. 2014],

H_λ [Zhou et al. 2010], and our RP^3_β. **(iii) Approximated/Sampled vertex ranking algorithms** $\hat{\mathrm{P}}^3_\alpha$, $\mathrm{R}\hat{\mathrm{P}}^3_\beta$, and $\hat{\mathrm{H}}_\lambda$. **(iv) Other algorithms:** MostPop (global item popularity), Random (random item ranking), weighted (WI-kNN) and unweighted (I-kNN) k-nearest neighbor item-based collaborative filtering using cosine distance as item similarity measure, and BPRMF [Rendle et al. 2009] (a recommender based on a latent factor model obtained with matrix factorization) – all available in MyMediaLite. To facilitate performance comparison, we also calculated the performance of the perfect recommender (Perfect) that places all test items of a user in random order at the top of the recommendation list.

Parameter Tuning. We empirically tune the parameters for parameterized algorithms to maximize the two accuracy metrics. For I-kNN and WI-kNN with MovieLens-M and iPlayer we tested neighborhood sizes $\mathrm{k} \in \{10, 50, 100, 150, 200\}$. For the BookCrossing I-kNN was tested for $\mathrm{k} \in \{10, 50, 100, 150, 200, 400, 800, 1600\}$ and for WI-kNN we additionally tested $\mathrm{k} = 3200$. Similarly, BPRMF was tested with the latent factors $\mathrm{d} \in \{10, 50, 100, 150, 200\}$ for Movie-Lens-M and BookCrossing. Due to computational limitations, BPRMF could only be tested with $\mathrm{d} \in \{10, 50\}$ for iPlayer.[4] For P^3_α, we tested values of $\alpha \in [-0.2, 4.5]$ in steps of 0.1. For RP^3_β, we tested values of $\beta \in [-0.2, 1.2]$ in steps of 0.1. H_λ was tested with values of $\lambda \in [0, 1]$ in steps of 0.1. The best performing parameters can be found in parentheses in the results Table II.

9.2. RP^3_β increases Accuracy and Diversity

The goal of the first set of experiments is to evaluate our re-ranking procedure RP^3_β. To that end we compare it with the other algorithms evaluated and especially explore its performance compared to P^3_α and H_λ.

As Table II shows, the RP^3_β re-ranking increases both accuracy and diversity for all datasets compared to its P^3 basis. Measured by AUC, RP^3_β is the most accurate algorithm for MovieLens-M and second most accurate algorithm after H_λ for iPlayer and BookCrossing. For Prec the results are less favorable: while the performance of RP^3_β is best for MovieLens-M and second best for iPlayer, WI-kNN, I-kNN, and H_λ clearly outperform RP^3_β for BookCrossing. This is possibly due to the lower number of average ratings per item, which may distort our boosting of low degree items.

[4]Other parameters for BPRMF: 30 stochastic gradient ascent iterations for training, no item bias, iteration length of 5, learning rate α of 0.05, regularization parameter for positive item factors of 0.0025, regularization parameter for negative item factors of 0.00025, and regularization parameter for user factors of 0.0025

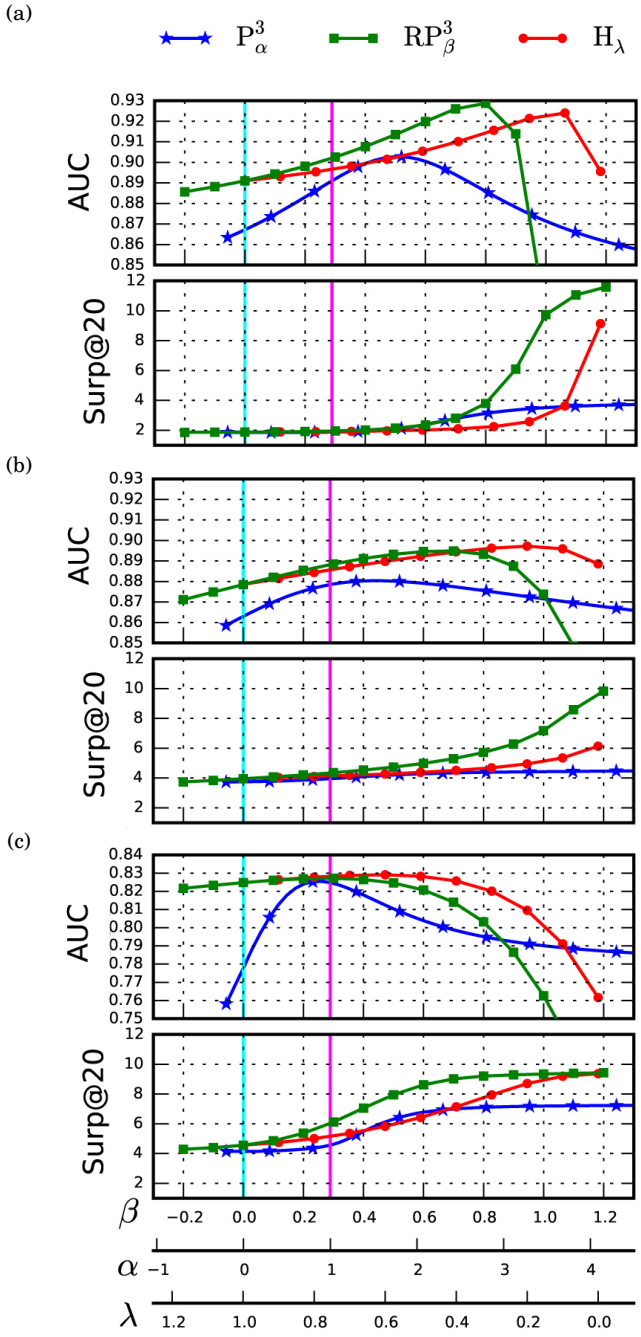

Fig. 2: AUC and Surp@20 performance of P^3_α, RP^3_β, and H_λ at different parameter values for different datasets: (a) MovieLens-M, (b) iPlayer, and (c) BookCrossing. The left vertical line (cyan) at $\beta = 0.0$, $\alpha = 0.0$, and $\lambda = 1.0$ indicates the parameter values where RP^3_β and H_λ give the same item ranking as P^3, and P^3_α the ranking of #3-Paths. The right vertical line (magenta) at $\alpha = 1.0$ indicates the parameter value where P^3_α gives the same item ranking as P^3.

	Recommender	AUC	Prec@20	GiniD@20	Pers@20	Surp@20
MovieLens-M	Perfect	1.0000	0.835	0.218	0.927	4.01
	RP^3_β ($\beta = 0.8$)	**0.9287**	0.341	0.172	0.941	**3.79**
	RP^3_β ($\beta = 0.7$)	**0.9260**	**0.359**	0.080	0.862	2.80
	H_λ ($\lambda = 0.1$)	**0.9240**	0.338	0.100	0.913	**3.63**
	H_λ ($\lambda = 0.2$)	0.9214	0.347	0.052	0.831	2.58
	BPRMF (d=50)	0.9211	0.324	0.189	0.952	3.22
	BPRMF (d=200)	0.9197	0.333	0.145	0.932	2.96
	WI-kNN (k=150)	0.9180	**0.353**	0.090	0.918	2.75
	WI-kNN (k=200)	0.9178	**0.354**	0.085	0.912	2.71
	I-kNN (k=150)	0.9138	0.283	**0.221**	**0.973**	3.62
	I-kNN (k=50)	0.9056	0.295	**0.204**	**0.968**	3.47
	P^3_α ($\alpha = 1.8$)	0.9028	0.259	0.027	0.644	2.13
	P^3_α ($\alpha = 1.5$)	0.9011	0.263	0.015	0.565	1.96
	P^3	0.8910	0.252	0.011	0.497	1.88
	L^+	0.8811	0.215	**0.218**	**0.971**	**4.22**
	#3-Paths	0.8672	0.234	0.010	0.449	1.86
	P^5	0.8600	0.217	0.009	0.410	1.84
	MostPop	0.8514	0.210	0.009	0.401	1.84
	Random	0.5018	0.015	0.900	0.994	6.33
iPlayer	Perfect	0.9618	0.120	0.068	0.172	8.21
	H_λ ($\lambda = 0.2$)	**0.8972**	**0.059**	0.251	0.805	4.94
	RP^3_β ($\beta = 0.7$)	**0.8949**	**0.059**	**0.327**	**0.848**	**5.29**
	WI-kNN (k=150)	**0.8911**	**0.058**	0.195	0.734	4.55
	P^3_α ($\alpha = 1.5$)	0.8804	0.051	0.139	0.617	4.13
	P^3	0.8785	0.049	0.108	0.567	3.95
	BPRMF (d=50)	0.8756	0.056	0.211	0.734	4.54
	#3-Paths	0.8630	0.043	0.077	0.490	3.75
	I-kNN (k=50)	0.8560	0.033	**0.340**	**0.867**	**6.11**
	I-kNN (k=10)	0.8056	0.046	**0.309**	**0.869**	**5.89**
	MostPop	0.7506	0.024	0.038	0.163	3.31
	Random	0.4950	0.004	0.954	0.968	8.01
BookCrossing	Perfect	1.0000	0.663	0.266	0.828	7.80
	H_λ ($\lambda = 0.6$)	**0.8291**	**0.080**	0.109	0.854	5.83
	H_λ ($\lambda = 0.5$)	**0.8283**	**0.082**	0.158	0.913	6.42
	RP^3_β ($\beta = 0.3$)	**0.8271**	0.071	0.154	0.876	6.11
	P^3_α ($\alpha = 0.9$)	0.8255	0.059	0.010	0.610	4.44
	P^3	0.8248	0.060	0.015	0.652	4.56
	P^3_α ($\alpha = 1.1$)	0.8235	0.060	0.026	0.703	4.74
	L^+	0.8234	0.033	**0.318**	**0.996**	**9.19**
	P^5	0.8056	0.042	0.002	0.271	4.09
	BPRMF (d=10)	0.7985	0.035	0.100	0.966	6.39
	WI-kNN (k=3200)	0.7825	0.060	0.118	0.955	6.91
	#3-Paths	0.7783	0.048	0.002	0.436	4.14
	BPRMF (d=200)	0.7735	0.048	0.109	0.965	5.87
	I-kNN (k=800)	0.7535	0.048	0.148	**0.976**	**7.38**
	MostPop	0.7180	0.034	0.001	0.111	3.95
	WI-kNN (k=50)	0.6542	**0.083**	**0.236**	0.975	**7.09**
	I-kNN (k=10)	0.5911	0.078	**0.178**	**0.978**	6.97
	Random	0.5010	0.001	0.748	0.999	8.57

Table II: Accuracy and diversity of all algorithms (ordered by decreasing AUC). Parameterized algorithms are represented by parameter values resulting in maximal AUC and Prec@20 performance. Top 3 numbers per metric highlighted (results from Perfect and Random recommender not considered).

Cooper *et al.* [Cooper et al. 2014] show that P^3_α improves accuracy over P^3. Our experiments confirm this claim but the accuracy improvements achieved with RP^3_β are even greater than with P^3_α for both AUC and Prec. Furthermore, at parameter values corresponding to maximum accuracy, RP^3_β achieves better GiniD, Pers, and Surp scores than P^3_α. This shows that RP^3_β gives a better trade-off between accuracy and diversity, i.e., at parameter values that achieve highest accuracy it produces more diverse results.

The results do not suggest a winner between RP^3_β and H_λ. In terms of AUC and Prec, RP^3_β has advantage over H_λ for MovieLens-M but not for iPlayer and BookCrossing. For BookCrossing the maximal achieved precision of H_λ is much better than that of RP^3_β. At parameter values corresponding to maximum accuracy, the diversity metric scores for RP^3_β are better for H_λ for MovieLens-M and iPlayer. Again, RP^3_β underperforms compared to H_λ on BookCrossing. Figure 2 graphs the AUC and Surp for P^3_α, RP^3_β, and H_λ for the whole parameter ranges. It shows that the maximally achieved Surp by RP^3_β is better (for MovieLens-M and iPlayer) or comparable (for BookCrossing) to H_λ. The plots for the other diversity measures show similar results but are omitted due to space considerations. Note that we measured the performance of H_λ only in the originally defined parameter interval ($\lambda \in [0, 1]$). We assume that the diversity performance of H_λ increases further for $\lambda < 0$ at the cost of accuracy.

We can conclude that the new method RP^3_β is a vertex ranking algorithm with top-class accuracy and diversity performance. Tuning of its parameter β allows the trade-off between recommendation accuracy and top-k long-tail item frequency to be controlled.

9.3. Performance of Vertex Ranking Algorithms

In this sub-section we compare the performance of vertex ranking algorithms to the others considered.

As Table II (page 23) shows, in accordance with [Cooper et al. 2014], P^3 is the most accurate algorithm among the measured parameter-free recommenders (MostPop, P^3, P^5, #3-Paths, and L^+). In particular, P^3 is more accurate than the computationally more expensive L^+ algorithm, which was found to be the most accurate algorithm in an earlier study [Fouss et al. 2005].

For AUC, the parameterized vertex ranking algorithms RP^3_β and H_λ outperform the non-vertex ranking recommendation algorithms I-kNN, WI-kNN, and BPRMF. For Prec, the scores of RP^3_β are high for the MovieLens-M dataset but low for the BookCrossing dataset; the opposite is true for H_λ. WI-kNN, the best performing non-vertex ranking algorithm, performs more consistently and archives comparable results to the best vertex ranking algorithm in terms of Prec.

The parameter free vertex ranking algorithms P^3, P^5, and #3-Paths clearly show lower diversity scores than I-kNN, WI-kNN, and BPRMF in all datasets. This is surprising considering the fact that I-kNN, WI-kNN, and BPRMF are more accurate for some of the datasets (e.g., MovieLens-M). Hence, the better diversity performance of the non-vertex recommenders is not explained by more randomness in their recommendations. Exploring the recommendation lists of P^3, P^5, and #3-Paths reveals that ranking is strongly biased by the item's degree (i.e., favoring blockbusters), resulting in rankings similar to MaxPop. The parameter free L^+ generates diverse recommendations at the cost of low Prec (worse than MostPop for BookCrossing). In terms of AUC it is almost as good as P^3.

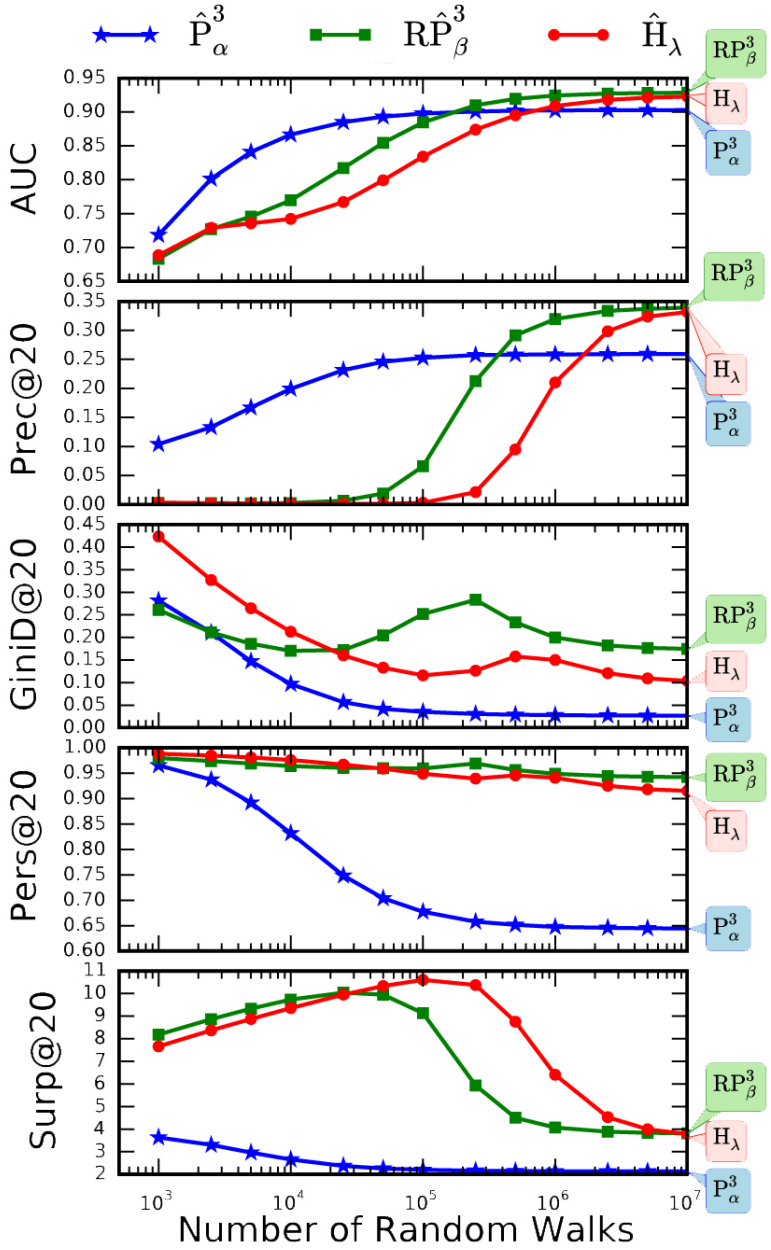

Fig. 3: Accuracy and diversity performance of the sampling algorithms $\hat{P}^3_\alpha$, $R\hat{P}^3_\beta$, and $\hat{H}_\lambda$ on MovieLens-M dataset for the parameter values of maximal AUC performance in dependency of the number of random walks per user. The annotations on the right-sided y-axis indicate the performance of the exact algorithms P^3_α, RP^3_β, and H_λ for the same parameter values.

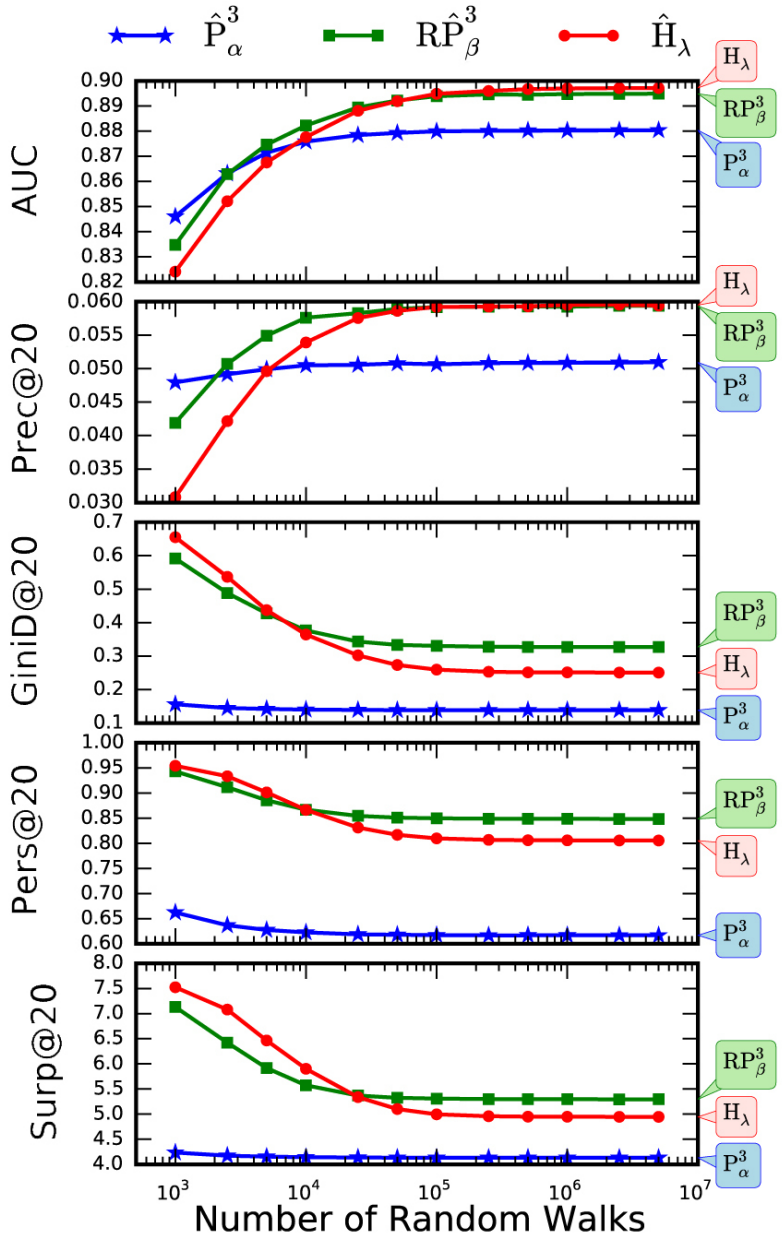

Fig. 4: Accuracy and diversity performance of the sampling algorithms $\hat{P}^3_\alpha$, $R\hat{P}^3_\beta$, and $\hat{H}_\lambda$ on iPlayer dataset for the parameter values of maximal AUC performance in dependency of the number of random walks per user. The annotations on the right-sided y-axis indicate the performance of the exact algorithms P^3_α, RP^3_β, and H_λ for the same parameter values.

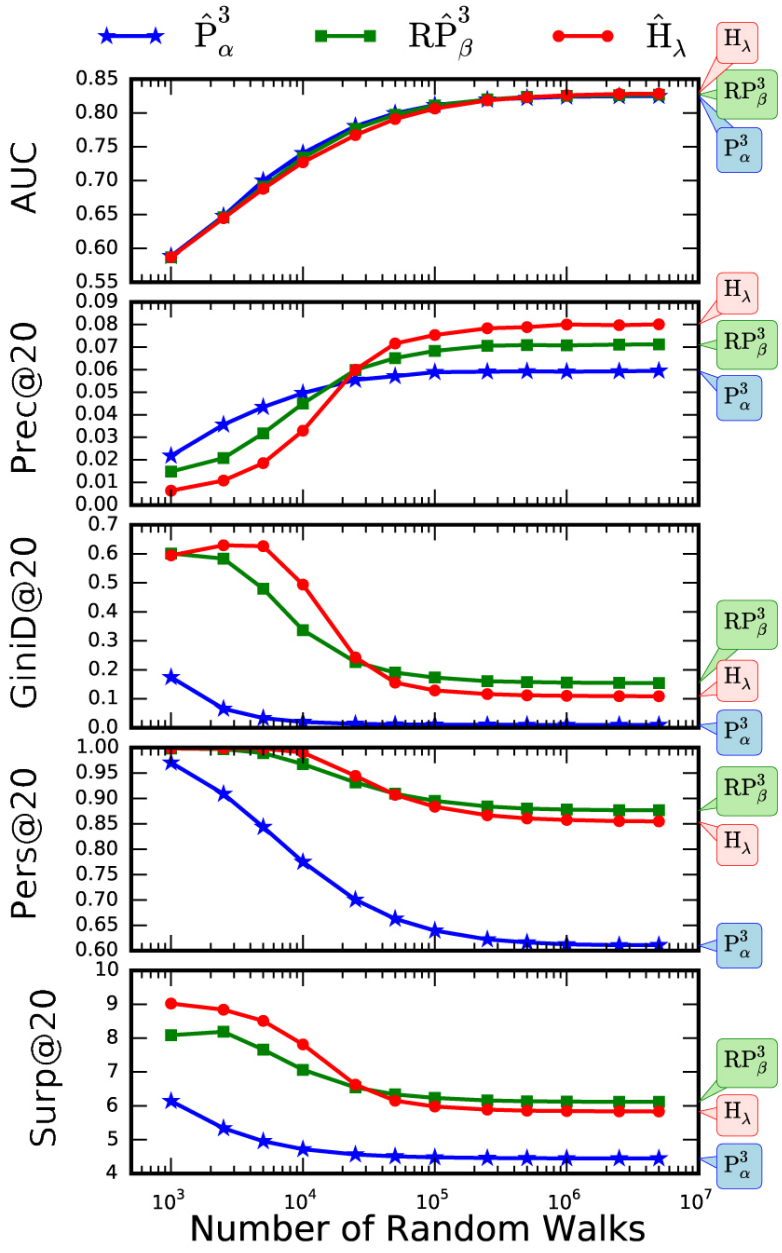

Fig. 5: Accuracy and diversity performance of the sampling algorithms $\hat{P}^3_\alpha$, $R\hat{P}^3_\beta$, and $\hat{H}_\lambda$ on BookCrossing dataset for the parameter values of maximal AUC performance in dependency of the number of random walks per user. The annotations on the right-sided y-axis indicate the performance of the exact algorithms P^3_α, RP^3_β, and H_λ for the same parameter values.

Parameterized vertex ranking algorithms provide, besides better accuracy, improved diversity compared to parameter free algorithms. Comparing the diversity performance of the most precise vertex (RP^3_β for MovieLens-M and iPlayer, H_λ for BookCrossing) and non-vertex (WI-kNN for all datasets) ranking recommendation algorithms reveals WI-kNN as the clear winner for BookCrossing: WI-kNN is not only slightly more precise than H_λ but also has higher diversity scores. For iPlayer RP^3_β is slightly more precise than WI-kNN and achieves higher diversity scores. No clear winner can be found for the MovieLens-M dataset: RP^3_β shows better precision and surprisal scores but WI-kNN succeeds in terms of GiniD and Pers performance.

Dataset	Recommender	AUC	Prec	GiniD	Pers	Surp
MovieLens-M (5 m walks)	$\hat{\mathrm{P}}^3_\alpha$ ($\alpha = 1.8$)	0.013	0.054	2.256	0.127	0.221
	$\mathrm{R}\hat{\mathrm{P}}^3_\beta$ ($\beta = 0.8$)	0.089	1.188	2.813	0.148	1.190
	$\hat{\mathrm{H}}_\lambda$($\lambda = 0.1$)	0.329	4.263	9.851	0.603	10.08
iPlayer (1 m walks)	$\hat{\mathrm{P}}^3_\alpha$ ($\alpha = 1.5$)	0.014	0.138	0.050	0.011	0.008
	$\mathrm{R}\hat{\mathrm{P}}^3_\beta$ ($\beta = 0.7$)	0.012	0.169	0.076	0.029	0.031
	$\hat{\mathrm{H}}_\lambda$($\lambda = 0.2$)	0.026	0.067	0.303	0.071	0.064
BookCrossing (1 m walks)	$\hat{\mathrm{P}}^3_\alpha$ ($\alpha = 0.9$)	0.173	0.605	1.089	0.441	0.074
	$\mathrm{R}\hat{\mathrm{P}}^3_\beta$ ($\beta = 0.3$)	0.237	0.324	1.103	0.247	0.186
	$\hat{\mathrm{H}}_\lambda$($\lambda = 0.6$)	0.359	0.125	1.527	0.436	0.287

Table III: Percentage of performance deviation between P^3_α, RP^3_β, and H_λ and $\hat{\mathrm{P}}^3_\alpha$, $\mathrm{R}\hat{\mathrm{P}}^3_\beta$, and $\hat{\mathrm{H}}_\lambda$ after 1 m or 5 m random walks per user for parameter values of maximal AUC performance.

9.4. Performance of Sampling Approximations

The goal of our second experiments is to investigate the performance of our sampling algorithms dependent on number of samples (i.e., number of random walks).

We determined the performance of our sampling algorithms $\hat{\mathrm{P}}^3_\alpha$, $\mathrm{R}\hat{\mathrm{P}}^3_\beta$, and $\hat{\mathrm{H}}_\lambda$ with parameter values of maximal AUC according to the non-sampling original algorithms whilst varying the number of random walks $N \in \{$ 1'000, 2'500, 5'000, 10'000, 25'000, 50'000, 100'000, 250'000, 500'000, 1 m, 2.5 m, 5 m$\}$ per user. Figure 3, 4, and 5 (corresponding to the datasets MovieLens-M, iPlayer, and BookCrossing respectively) show the rate of convergence as well as the performance of the exact algorithms as indicated by the callouts near right edge of each graph. As expected the sampled algorithms' performance converge to that of the exact ones with increasing N. To illustrate the closeness of the results we computed the percentage deviation $d = (|m - \hat{m}|) * 100/m$ between the sampling procedures' $\hat{m}$ and exact calculations' m performance metrics for 5 million random walks for MovieLens-M and 1 m random walks for iPlayer and BookCrossing. The results of this procedure, listed in Table III (Page 28), show that the sampled algorithms usually deviate less than 1% from the exact ones, less than 3% in all cases but for $\hat{\mathrm{H}}_\lambda$ for MovieLens-M. Despite the greater number of random walks, d is greater for the MovieLens-M dataset than for the iPlayer or BookCrossing datasets. We hypothesize that this is due to the greater number of distinct paths of length three starting at a given user existing in the graph G for MovieLens-M dataset as indicated

by the high average vertex degree of 71.8 (compared to iPlayer: 7.1, BookCrossing: 11.6).

Furthermore, Figures 3- 5 clearly indicates that $\hat{\mathrm{P}}^3_\alpha$ requires less samples to converge than $\mathrm{R}\hat{\mathrm{P}}^3_\beta$, which in turn converges faster than $\hat{\mathrm{H}}_\lambda$. Since these algorithms can be computed using Algorithm 1 and differ only in c_{rw}, we hypothesize that c_{rw} controls the efficiency of sampling. As a result $\hat{\mathrm{P}}^3_\alpha$ is the most accurate sampling algorithm for small values of N. For slightly greater N, $\mathrm{R}\hat{\mathrm{P}}^3_\beta$ is more accurate than $\hat{\mathrm{P}}^3_\alpha$ in the MovieLens-M and iPlayer datasets. If we increase N even further, $\hat{\mathrm{H}}_\lambda$ becomes the most accurate recommender for the iPlayer and BookCrossing dataset.

Considering recommendation accuracy, diversity, and the sample size required to obtain acceptable accuracy, our results suggest the following: On data with moderate sparsity and balanced user and item degrees (MovieLens-M) one should use $\hat{\mathrm{P}}^3_\alpha$ if computing resources are scarce, i.e., $N < 250'000$, because of the algorithm's better precision and otherwise $\mathrm{R}\hat{\mathrm{P}}^3_\beta$ which provides best accuracy and diversity (at comparable level of accuracy). For more sparse data with more ratings per item than per user (iPlayer), $\mathrm{R}\hat{\mathrm{P}}^3_\beta$ is probably the best choice since it reaches almost the maximal accuracy but gives better diversity (at comparable level of accuracy) and converges quicker than $\hat{\mathrm{H}}_\lambda$. For a sparse dataset with an average item degree that is greater than the average user degree (BookCrossing) $\hat{\mathrm{H}}_\lambda$ is clearly the best choice given that computing resources are plenty ($N > 25'000$), since it gives better precision and diversity (at comparable level of accuracy). In the case of limited computing power however, the choice is not obvious due to the poor accuracy of $\mathrm{R}\hat{\mathrm{P}}^3_\beta$ and $\hat{\mathrm{H}}_\lambda$ and very poor diversity of $\hat{\mathrm{P}}^3_\alpha$.

9.5. The Effect of Caching on Sample Size

In the previous sub-section we explored the convergence speeds and diversity of our approximate algorithms $\mathrm{R}\hat{\mathrm{P}}^3_\beta$, $\hat{\mathrm{H}}_\lambda$, and $\hat{\mathrm{P}}^3_\alpha$. As discussed in Section 6, the performance of the approximate algorithms can be further improved with the caching of partial walks. Specifically, we explained how the caching of partial 1-hop walks from user to item vertices allows the calculation of multiple 3-hop samples with one 2-hop walk. In this sub-section we explore the magnitude of this saving.

To ascertain the increase of samples per actual walk though caching, we ran 30'000 actual random walks per user on Movielens-M and 10'000 actual random walks per user on BookCrossing and counted the resulting number of samples per vertex.[5] Figure 6 graphs the effect of caching for the two datasets by plotting the number of samples obtained per user vertex (the logarithmic y-axis) vs. the degree of the vertex (x-axis).

As indicated in the top part of the Figure, caching allowed the equivalent 47'506 samples on average per user vertex with for Movielens-M for the started 30'000 walks from each vertex. High-degree vertices benefit significantly from caching with a maximum of 516'706 samples. Where caching has no effect the algorithm can 'only' provide 30'000 samples per user (i.e., one sample per walk). The effect for BookCrossing (bottom graph) is comparable (10'000 walks resulting in a maximum of 872'119 and an average of 15'864 samples per user).

In conclusion, on average caching leads to 1.5 times more samples than walks. As expected, high-degree vertices profit significantly more form caching that lower degree ones. Hence, if used adaptively, this approach could further cut down on computation time to convergence.

[5]The proprietary iPlayer dataset was not available for the caching and update evaluations.

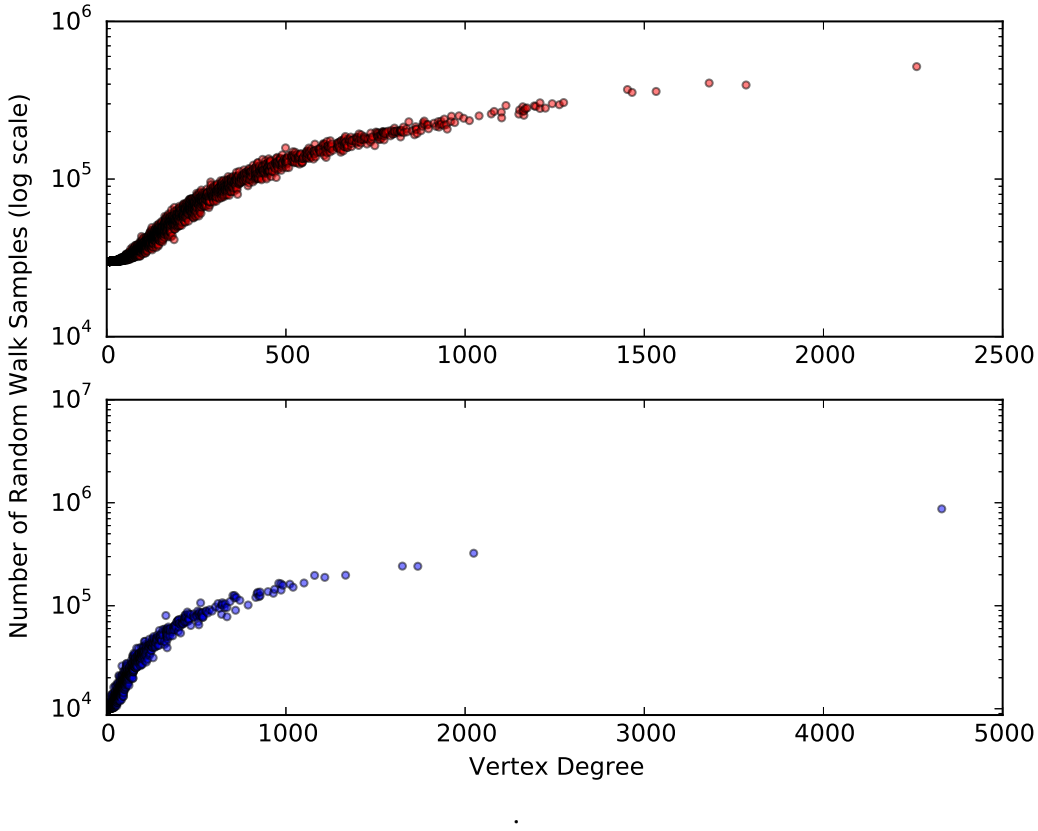

Fig. 6: Caching random walk segments has the effect of generating more samples for high degree nodes. Depending on the dataset, nodes with high degree receive between 10 and 80 times more samples.
The top plot shows the varying number of samples for users in the MovieLens-M dataset. Based on 30'000 walks from each user vertex, we received between 30'000 and 516'706 samples (average = 47'506).
The bottom plot shows the distribution for the BookCrossing dataset, where we started with 10'000 walks from each vertex resulting between 10'000 and 872'119 samples (average = 15'864).

9.6. Updating Recommendations when Adding Edges at Interactive Speeds

One of the core claims of our paper is that our updating approach from Section 7 speeds up the re-computation of recommendations to a degree that it allows the use of on-line computed recommendation lists for interactive applications. As discussed in the usability literature (see chapter 5 in [Nielsen 1993])[6] there are three limits for response times. Users experience speeds up to $\frac{1}{10}sec$ as instantaneous, up $1sec$ as non-disruptive to the flow of thought, and need to be provided with feedback within $10secs$ to keep their attention. Consequently, in order for our recommendations to be useful in interactive applications we should provide them within $\frac{1}{10}$ of a second.

[6]Also available at: https://www.nngroup.com/articles/response-times-3-important-limits/

To ascertain if our updating mechanism achieves this goal we took both the MovieLens-M and the BookCrossing dataset and removed 10'000 random edges from the graph. While doing so, we made sure that each vertex had at least one edge still remaining after the removal of edges to avoid any newly unconnected vertices. Then we calculated the initial recommendation lists on the graph with 10'000 edges removed.

To find the actual update times we then added the previously removed 10'000 one at a time and measured the time it took to update the partial quantities. Obviously, the time varies according to the degree of the neighboring vertices of the user-item graph around the added edge. The resulting distribution of update times per user vertex is graphed in Figure 7. As the figure clearly shows, all update times were well within interactive speeds. For MovieLens-M, the fastest updates are at 10.8 μs, the slowest at 12 ms, with an average at 1.99 ms. For BookCrossing, we find that the fastest updates are at 10.3 μs, the slowest at 4 ms, with an average at 0.1 ms. We can, hence, conclude that our update approach can provided updated recommendations at interactive speeds. All these times do not include other constant operations like the ranking of the updated recommendation scores, which took about 15-20ms in our experiments—a time that could be significantly improved if the need arose.

Note, that these times will not increase with the size of the item-recommendation graph. The 'only' element affecting the update time is the degree of the vertices neighboring (2-hop for users and 1-hop for items) the added edge. In future work we hope to explore the precise interaction between the degree of the neighboring vertices and update speed analytically to be able to provide upper bounds for the update time. Such a prediction would allow deciding when to wait for an update and when to use a 'stale' recommendation list.

10. CONCLUSIONS AND FUTURE WORK

In this paper, we studied accuracy and diversity of vertex ranking algorithms using random walk sampling techniques and thereby bring together three streams of earlier presented work. Specifically, we introduced RP^3_β, a novel graph random walk based recommendation algorithm based on a re-ranking of P^3 that gives better recommendation accuracy and diversity than previously proposed vertex ranking algorithms. We showed that re-ranking improves the accuracy performance over P^3 and its parameterized version P^3_α and pushes "wallflowers", i.e., long-tail items, closer to the top of the recommendation list. Our method is also competitive with another graph-based recommender H_λ that optimizes the accuracy diversity trade-off. We also showed that RP^3_β is competitive with traditional algorithms.

Additionally, we presented efficient and scalable random walk sampling implementations of these three algorithms. We showed empirically that these algorithms converge to their exact counterparts with increasing number of samples. The sampling procedures have the favorable property of being anytime algorithms: a recommendation list of low accuracy can be generated after a short processing time, while longer computations, i.e., gathering more random walk samples, improve the accuracy of the recommendation list. In future work we hope to investigate the sensitivity of the convergence of the sampling algorithms to domain characteristics and further explore convergence behavior for different datasets and algorithms.

Furthermore, we introduced a a caching approach for our sampling algorithms that increases the sampling efficiency by 1.5 on average. If used adaptively, this approach could further cut down on computation time to convergence—an investigation we hope to address in future work.

Last and highly important, we introduce an update technique for our sampled approaches. For the two datasets investigated, this update approach allows to refresh

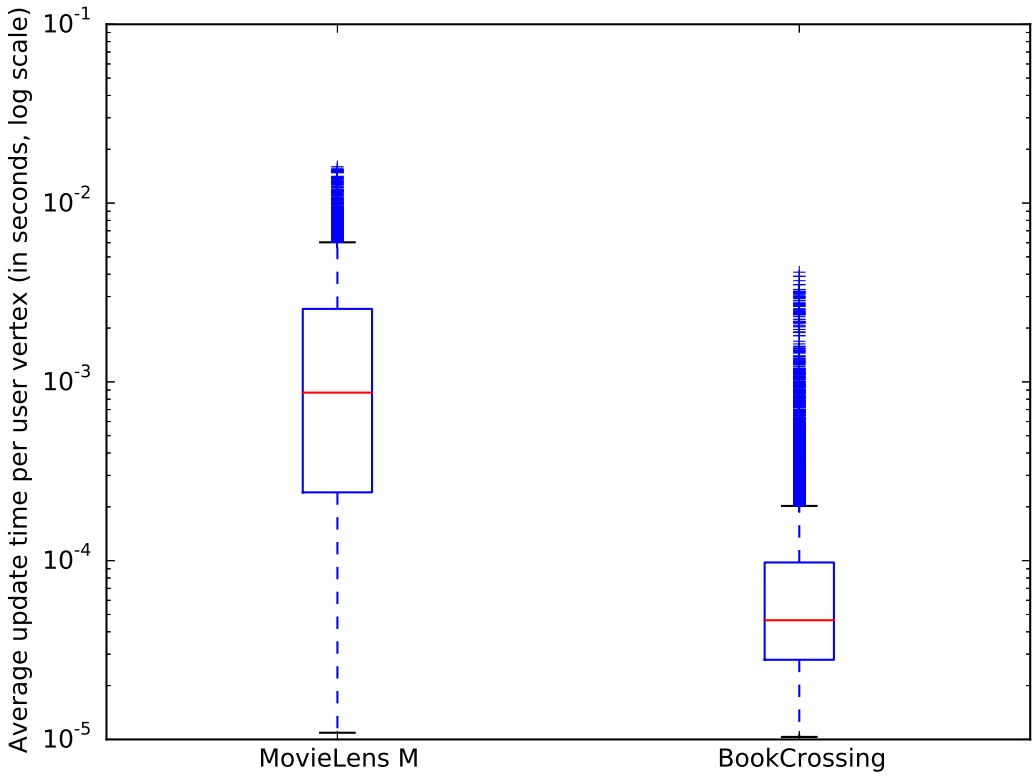

Fig. 7: Distribution of average update time per vertex for the 10'000 edge additions in the two datasets.
The box plot graphs the median, the boundaries of the inter-quartile range, 1.5 **times above/below the inter-quartile range, and extrema for the average per-vertex update times for the 10'000 updates.**

recommendations when new user-item relations are introduces in the sub-$\frac{1}{10}sec$-range, making our algorithms suitable for interactive applications. An outstanding element, is the analytic computation of an upper bound of update times based on the user-item graph characteristics in the neighborhood of the added edge—an investigation we hope to entertain in future work.

The goal of this paper was providing accurate, diverse, and scalable recommendations for interactive applications, where updates are considered on-line as new information becomes available. For the datasets investigated, our results indicate that the goal of fast, accurate, and surprising recommendations could be reached with vertex ranking algorithms using random walk sampling. For two of the three datasets we could ascertain that the time to compute these recommendations can be reduced via caching approaches allowing updates at interactive speeds. We think, that these findings pave the way to interactively updated recommendations in intelligent interactive applications.

Acknowledgements

The authors would like to thank the Hasler Foundation for generously supporting this work under grant #11072 and the anonymous reviewers for their comments, which significantly helped to improve this paper.

REFERENCES

Gediminas Adomavicius and YoungOk Kwon. 2011. Maximizing aggregate recommendation diversity: A graph-theoretic approach. In *Proceedings of the 1st International Workshop on Novelty and Diversity in Recommender Systems (DiveRS 2011)*. Citeseer, 3–10.

Gediminas Adomavicius and YoungOk Kwon. 2012. Improving aggregate recommendation diversity using ranking-based techniques. *Knowledge and Data Engineering, IEEE Transactions on* 24, 5 (2012), 896–911.

Charu C Aggarwal, Joel L Wolf, Kun-Lung Wu, and Philip S Yu. 1999. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 201–212.

David Aldous and Jim Fill. 2002. Reversible Markov chains and random walks on graphs. (2002).

Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment* 4, 3 (2010), 173–184.

Shumeet Baluja, Rohan Seth, D Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. 2008. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th International Conference on World Wide Web*. ACM, 895–904.

Pavel Berkhin. 2006. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics* 3, 1 (2006), 41–62.

Toine Bogers. 2010. Movie recommendation using random walks over the contextual graph. In *Proceedings of the 2nd International Workshop on Context-Aware Recommender Systems*.

Matthew Brand. 2005. A Random Walks Perspective on Maximizing Satisfaction and Profit. In *SDM*. SIAM, 12–19.

John S Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 43–52.

Steve Chien, Cynthia Dwork, Ravi Kumar, Daniel R Simon, and D Sivakumar. 2004. Link evolution: Analysis and algorithms. *Internet mathematics* 1, 3 (2004), 277–304.

Fabian Christoffel, Bibek Paudel, Chris Newell, and Abraham Bernstein. 2015. Blockbusters and Wallflowers: Speeding up Diverse and Accurate Recommendations with Random Walks. In *9th ACM Conference on Recommender Systems RecSys 2015*, Jennifer Golbeck and Giovanni Semeraro (Eds.). ACM Press, New York, NY, USA. DOI:http://dx.doi.org/10.1145/2792838.2800180

Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. 2014. Random Walks in Recommender Systems: Exact Computation and Simulations. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*. International World Wide Web Conferences Steering Committee, 811–816.

Paolo Cremonesi, Franca Garzotto, Sara Negro, Alessandro Vittorio Papadopoulos, and Roberto Turrin. 2011. Looking for "good" recommendations: A comparative evaluation of recommender systems. In *INTERACT 2011*. Springer, 152–168.

Daniel Fleder and Kartik Hosanagar. 2009. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management science* 55, 5 (2009), 697–712.

Daniel M Fleder and Kartik Hosanagar. 2007. Recommender systems and their impact on sales diversity. In *Proceedings of the 8th ACM conference on Electronic commerce*. ACM, 192–199.

Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics* 2, 3 (2005), 333–358.

Francois Fouss, Alain Pirotte, and Marco Saerens. 2005. A novel way of computing similarities between nodes of a graph, with application to collaborative recommendation. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 550–556.

Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A free recommender system library. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 305–308.

David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (1992), 61–70.

Daniel G Goldstein and Dominique C Goldstein. 2006. Profiting from the long tail. *Harvard Business Review* 84, 6 (2006), 24–28.

Marco Gori, Augusto Pucci, V Roma, and I Siena. 2007. ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines.. In *IJCAI*, Vol. 7. 2766–2771.

Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 505–514.

Taher H Haveliwala. 2002. Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*. ACM, 517–526.

Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 230–237.

Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 5–53.

Zan Huang, Hsinchun Chen, and Daniel Zeng. 2004. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 116–142.

Mohsen Jamali and Martin Ester. 2009. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 397–406.

Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 538–543.

Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*. ACM, 271–279.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

Amy N Langville and Carl D Meyer. 2006. Updating Markov chains with an eye on Google's PageRank. *SIAM J. Matrix Anal. Appl.* 27, 4 (2006), 968–987.

Sangkeun Lee, Sungchan Park, Minsuk Kahng, and Sang-goo Lee. 2012. Pathrank: a novel node ranking measure on a heterogeneous graph for recommender systems. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. ACM, 1637–1641.

Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. Indianapolis, IN, USA, 135–146. DOI:http://dx.doi.org/10.1145/1807167.1807184

Sean M McNee, John Riedl, and Joseph A Konstan. 2006. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems*. ACM, 1097–1101.

Jakob Nielsen. 1993. *Usability Engineering*. Morgan Kaufmann Publishers. https://www.nngroup.com/articles/response-times-3-important-limits/

Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. 2015. Efficient PageRank Tracking in Evolving Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 875–884.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: bringing order to the web. (1999).

Eli Pariser. 2011. *The filter bubble: What the Internet is hiding from you*. Penguin UK.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. ACM, 175–186.

Ruslan Salakhutdinov and Andriy Mnih. 2008. Bayesian Probabilistic Matrix Factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine learning*. ACM, 880–887.

Ruslan Salakhutdinov and Andriy Mnih. 2011. Probabilistic Matrix Factorization. In *NIPS*, Vol. 20. 1–8.

Purnamrita Sarkar, Andrew W Moore, and Amit Prakash. 2008. Fast incremental proximity search in large graphs. In *Proceedings of the 25th International Conference on Machine Learning*. ACM, 896–903.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*. ACM, 285–295.

Philip Stutz, Abraham Bernstein, and William Cohen. 2010. Signal/Collect: Graph Algorithms for the (Semantic) Web. In *Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part I (ISWC '10)*. Shanghai, China, 764–780. http://dl.acm.org/citation.cfm?id=1940281.1940330

Philip Stutz, Daniel Strebel, and Abraham Bernstein. 2016. Signal/Collect: Processing Large Graphs in Seconds. *Semantic Web Journal* 7 (2016), 139–166.

Saúl Vargas and Pablo Castells. 2011. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 109–116.

Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. 2010. Temporal recommendation on graphs via long-and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 723–732.

Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. 2012. Challenging the long tail recommendation. *Proceedings of the VLDB Endowment* 5, 9 (2012), 896–907.

Liyan Zhang, Kai Zhang, and Chunping Li. 2008. A topical pagerank based algorithm for recommender systems. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 713–714.

Yuan Cao Zhang, Diarmuid Ó Séaghdha, Daniele Quercia, and Tamas Jambor. 2012. Auralist: introducing serendipity into music recommendation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*. ACM, 13–22.

Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-Rank: a comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM, 553–562.

Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. 2010. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences* 107, 10 (2010), 4511–4515.

Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*. ACM, 22–32.

Received February 2007; revised March 2009; accepted June 2009